Date de dernière modification *: 30/6/2015*

Annexe **18**

Analyse conceptuelle du domaine d'application

Compléments

Cette annexe comprend des matériaux complémentaire au chapitre 18. En particulier, elle développe plus avant les problématiques de **correction d'un schéma**, de **normalisation d'un schéma**, d'**évaluation d'un schéma par prototypage** et d'**intégration de schémas**.
En outre, elle comprend un jeu d'**exercices**, dont la plupart sont accompagnés d'une solution ou de conseils de résolution.

A18.1 PROCESSUS DE CORRECTION D'UN SCHÉMA

<complément de la section 18.6>

Un schéma conceptuel est un document généralement volumineux et complexe, et à ce titre susceptible de contenir des **erreurs**. Il en existe de plusieurs types, de sorte que l'analyste n'aura souvent que l'embarras du choix !

Les *maladresses de formulation* (les constructions du schéma sont correctes mais mal exprimées) seront traitées dans la section A18.2, consacrée à la *normalisation* du schéma. Les *erreurs sémantiques* (le schéma ne représente pas correctement les exigences des utilisateurs) seront traitées à la section 18.8, consacrée à l'*évaluation* du schéma. La présente section étudie un type d'erreurs identifiable formellement et

qu'il est indispensable de corriger avant de procéder à la normalisation et à la validation[1].

On distinguera trois phénomènes distincts : les constructions *syntaxiquement incorrectes*, les constructions *non satisfiables* et les constructions *incohérentes*.

A18.1.1 Constructions syntaxiquement incorrectes

Une erreur syntaxique affecte une construction dont la formulation ne respecte pas les règles de constitution propres au modèle choisi (chapitres 11 et 15). En principe, ce type d'erreurs est plutôt rare lorsque le schéma est construit à l'aide d'un AGL, lequel effectue en temps réel ou à la demande de l'analyste des vérifications de forme. Des erreurs peuvent cependant subsister suite à une utilisation abusive de l'atelier, au laxisme de ce dernier ou lorsque l'analyste utilise pour construire son schéma un logiciel de dessin généraliste tel que *Visio* ou *Powerpoint*. Le schéma de la figure A18.1 comporte quelques erreurs syntaxiques classiques : le type d'associations affilié n'a qu'un seul rôle, qui d'ailleurs est affecté d'une contrainte de cardinalité incorrecte, deux types d'entités portent le même nom, la composition de l'identifiant de ORDRE est invalide (pas d'attribut multivalué dans un identifiant multi-composants), la contrainte de coexistence de TITRE (droite) comporte un composant obligatoire, l'identifiant primaire de ORDRE comporte un composant facultatif, l'identifiant de acheter est invalide (doit inclure au moins un rôle).

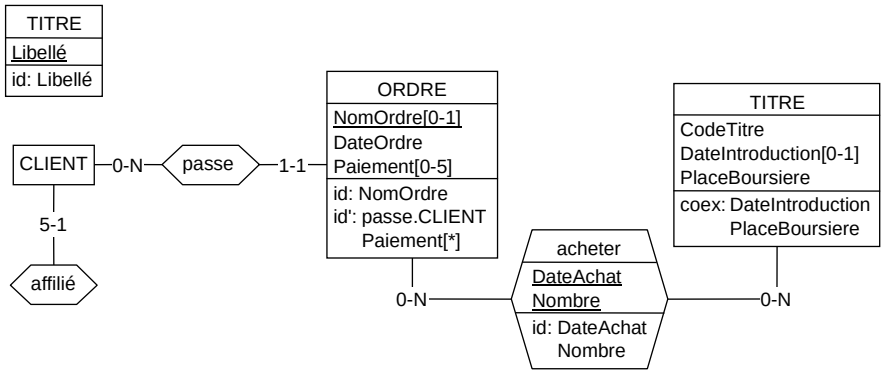


Figure A18.1 - Le jeu des 7 erreurs (syntaxiques)

Il existe bien d'autres erreurs de ce type qui dérivent du non respect des propriétés du modèle utilisé. Le relecture attentive du chapitre 16 permettra d'en repérer un grand nombre. La figure A18.2 en suggère trois, liées aux structures de relations *is-a*.

- *Schéma de gauche* : un type d'entités ne peut être un sous-type de lui-même ; en d'autres termes, le graphe des relations *is-a* ne peut comprendre de circuits[2].

1. Sur la détection et la correction des erreurs de cohérence, on consultera par exemple [Hartmann, 2001].

- *Schéma de droite* : on ne peut définir plus d'un identifiant primaire pour un type d'entités ; deux attributs de types d'entités d'une même branche d'une hiérarchie *is-a* ne peuvent porter le même nom[3].

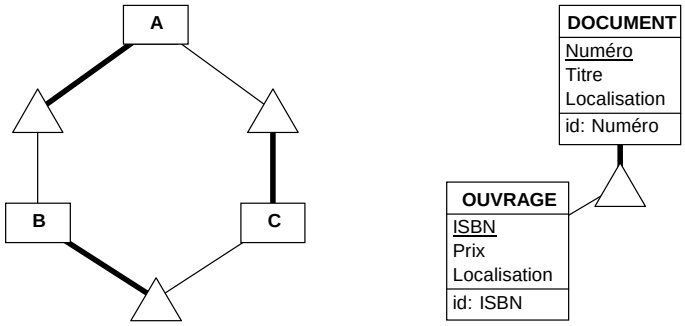


Figure A18.2 - Trois problèmes dans une structure de relations *is-a*

Correction

Une erreur syntaxique empêchant tout traitement ultérieur du schéma, elle doit impérativement être corrigée. Le *repérage* de ces erreurs peut être automatisé mais leur correction nécessite la compréhension du domaine d'application.

A18.1.2 Constructions non satisfiables et incohérentes

La figure 16.21 comprend deux schémas que nous avons qualifiés de *problématiques*. Dans chacun d'eux, la population du type d'entités D est vide par construction. Quelles que soient notre ingéniosité et notre ténacité, et quelle que soit la technologie de mise en œuvre, il nous sera impossible de créer une seule entité D qui respecte les contraintes de ces schémas. Aucune instance de D ne pouvant satisfaire ces deux schémas, on dira que D est une construction *non satisfiable*.

Une construction d'un schéma (type d'entités, attribut, type d'associations) est dite **satisfiable** lorsqu'elle admet des instances *non vides* (et *finies*). Elle est **non satisfiable** lorsqu'il n'est pas possible d'en créer des instances qui respectent toutes ses contraintes. La non-satisfiabilité découle de contraintes trop fortes, le plus souvent dues à une erreur d'analyse. Nous examinerons trois grandes classes de constructions non satisfiables.

2. Curieusement, les types d'entités de ce schéma sont parfaitement *satisfiables* dès qu'on admet que les trois types d'entités sont identiques : mêmes composants et même population.
3. Cette structure pourrait être perçue comme correcte mais non normalisée. En effet, si les deux attributs ont même sémantique, alors l'attribut inférieur correspond à une *redondance structurelle d'instances*, traitée en A18.2.7. Cette construction se rencontrera notamment dans le processus d'intégration (18.9).

a) Sous-types non satisfiables

Le problème du sous-type vide par construction a été décrit à la section 16.9.5. Il peut se poser dès qu'un type d'entités possède plus d'un surtype. La figure A18.3 illustre les deux cas représentatifs. Dans le premier schéma EPARGNE-ASSURANCE est non satisfiable en raison du principe selon lequel les populations de deux types d'entités sans surtype commun sont *disjointes par définition*. Tout sous-type qui leur serait commun serait non satisfiable. Le second schéma pose le même problème, les deux sous-types VEHICULE TERRESTRE et VEHICULE AQUATIQUE étant *disjoints par déclaration*.

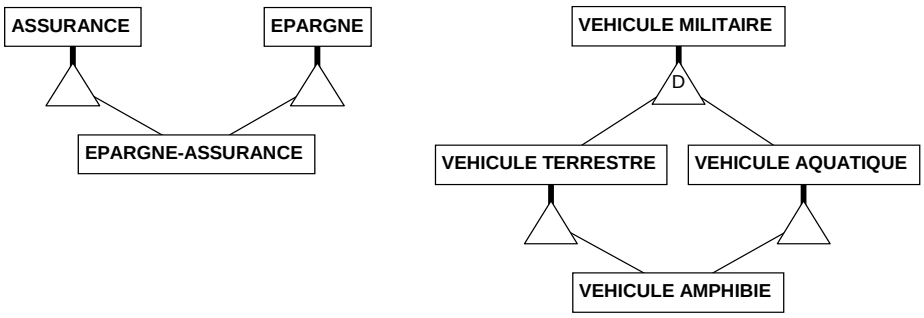


Figure A18.3 - Les sous-types EPARGNE-ASSURANCE et VEHICULE AMPHIBIE sont non satisfiables

La question de la satisfiabilité dans les hiérarchies *is-a* a été étudiée par plusieurs auteurs. Voir [Balaban, 2006] par exemple comme point d'entrée.

Correction

Sauf dans des cas très particuliers[4], une construction non satisfiable résulte d'une erreur d'analyse. Les types d'entités non satisfiables sont assez faciles à identifier. On appliquera pour ce faire les règles suivantes, qui précisent les observations de la section 16.9.5. On se place dans l'hypothèse de répartition simple (section 16.9.7).

Deux définitions :

- Deux types d'entités sont *structurellement disjoints* si leurs populations sont toujours disjointes quelles que soient les populations des autres types d'entités du schéma.
- Un type d'entités est *structurellement vide* si sa population est toujours vide quelles que soient les populations des autres types d'entités du schéma.

Par ailleurs, on sait que :

- Deux types d'entités qui n'ont pas de surtype sont *structurellement disjoints*.

4. Par exemple, lorsqu'il est prévu que le schéma évolue dans l'avenir et que la contrainte aujourd'hui excessive soit relâchée plus tard, permettant alors la création d'entités du type non satisfiable.

- Deux types d'entités dont un surtype direct commun est soumis à une contrainte de disjonction sont *structurellement disjoints*.

On en déduit les règles de satisfiabilité :

1. Si deux types d'entités sont *structurellement disjoints*, tout sous-type de l'un est *structurellement disjoint* de l'autre.
2. Tout sous-type d'un type d'entités *structurellement vide* est *structurellement vide*.
3. Tout type d'entités dont deux surtypes directs sont *structurellement disjoints* est *structurellement vide*.

Ces règles doivent être appliquées récursivement pour traiter des hiérarchies de hauteur quelconque. On corrigera ces constructions par la relaxation des contraintes.

b) Contraintes d'existence non satisfiables

Considérons un type d'entités auquel sont associés des composants facultatifs (attributs et/ou rôles) et faisant l'objet de contraintes d'existence (section 16.11.3). Lorsque ces contraintes sont mal définies, on peut conclure qu'il est impossible de donner une valeur à certains de ces composants ou même qu'il n'existe aucune configuration de ces composants satisfaisant ces contraintes. Dans le premier cas, ces composants sont non satisfiables et dans le second cas, c'est le type d'entités tout entier qui est non satisfiable. La figure A18.4 illustre ces deux situations.

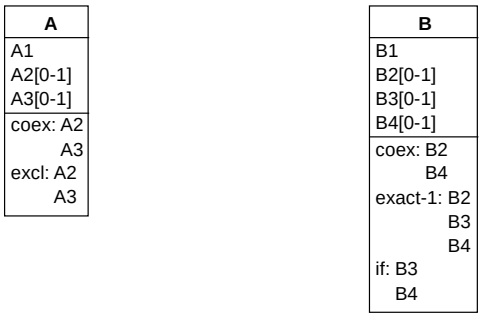


Figure A18.4 - Contraintes d'existence non satisfiables

Pour étudier ces contraintes, nous allons utiliser une table de vérité telle que celle de la section 16.11.3. Observons d'abord que la liste des contraintes définit une expression booléenne équivalente à leur conjonction. Par exemple, les contraintes du schéma de gauche correspondent globalement à l'expression :

$$\text{cond}(A2, A3) = (\text{coex: } A2, A3) \wedge (\text{excl: } A2, A3)$$

Nous pouvons alors construire la table de vérité calculant les valeurs de cette expression :

présence		*contraintes*		*résultat*
A2	**A3**	**coex: A2, A3**	**excl: A2, A3**	**cond(A2, A3)**
0	0	1	1	**1**
0	1	0	1	0
1	0	0	1	0
1	1	1	0	0

Il n'existe qu'une seule configuration satisfaisant aux deux contraintes : l'absence de valeur pour A2 et A3. Ces deux attributs sont donc *non satisfiables*. Il est possible de créer des entités A pourvu qu'on n'assigne pas de valeurs à A2 et à A3.

Étudions de la même manière le schéma de droite. L'expression globale s'écrit :

$$\text{cond(B2, B3, B4)} = \text{(coex: B2, B4)} \wedge \text{(exact-1: B2, B3, B4)} \wedge \text{(if: B3, B4)}$$

La table de vérité pour ce schéma est la suivante :

présence			*contraintes*			*résultat*
B2	**B3**	**B4**	**coex: B2, B4**	**exact-1: B2, B3, B4**	**if: B3, B4**	**cond(B2, B3, B3)**
0	0	0	1	0	1	0
0	0	1	0	1	1	0
0	1	0	0	1	0	0
0	1	1	0	0	1	0
1	0	0	0	1	1	0
1	0	1	1	0	1	0
1	1	0	0	0	0	0
1	1	1	1	0	1	0

Ici, la conclusion est plus radicale : il n'existe **aucune** configuration des trois attributs qui satisfasse simultanément les trois contraintes ! En d'autres termes, le type d'entités B n'est pas satisfiable.

Correction

Les problèmes décrits résultent d'erreurs d'analyse qui doivent être corrigées. On évaluera pour chaque type d'entités la satisfiabilité des contraintes d'existence selon la procédure illustrée précédemment. En cas de problème, on corrigera le schéma de manière qu'il représente plus fidèlement le domaine d'application.

c) Cardinalités non satisfiables dans un cycle de types d'associations

Certaines valeurs des contraintes de cardinalité de types d'associations rendent impossible la création d'entités impliquées dans des associations de ces types. Les trois schémas de la figure A18.5 illustrent le phénomène.

Analysons le premier schéma selon les propriétés étudiées à la section 16.8.6. Soient $\mathbf{N}_A$ et $\mathbf{N}_B$ les nombres d'entités des types A et B respectivement. On a les relations suivantes :

$$\mu_{RA} \geq \mathbf{2};\ \mu_{RB} \leq \mathbf{1}$$

$$\mu_{SA} \leq \mathbf{1};\ \mu_{SB} = \mathbf{1}$$

On calcule la taille de la population de B selon le type d'associations R :

$$\mathbf{N}_B = \mathbf{N}_A \times (\mu_{RA} / \mu_{RB}) \geq \mathbf{2}\,\mathbf{N}_A$$

De même, on calcule la taille de la population de A selon le type d'associations S :

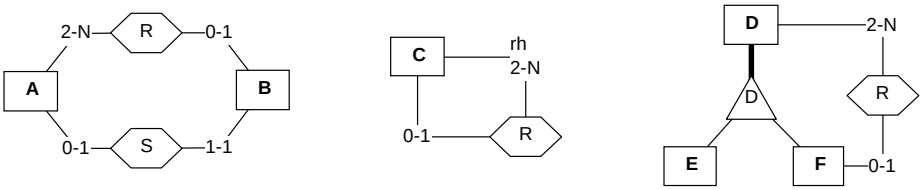


Figure A18.5 - Cardinalités non satisfiables

$$\mathbf{N}_A = \mathbf{N}_B \times (\mu_{SB} / \mu_{SA}) \geq \mathbf{N}_B$$

En remplaçant $\mathbf{N}_A$ par $\mathbf{N}_B$ dans la première expression, il vient :

$$\mathbf{N}_B \geq \mathbf{2}\,\mathbf{N}_B$$

Cette inégalité n'est vérifiée que pour $\mathbf{N}_B = 0$. On en conclut que ni B ni A ne sont satisfiables. Les deux autres schémas seront analysés selon le même procédé. On en conclut que les types d'entités C, D, E et F ne sont pas satisfiables.

L'exemple de la figure A18.6 ressemble au précédent, mais relève d'un phénomène différent puisque la contrainte de cardinalité posera un problème, non pas pour PERSONNEL et EXPERT, mais pour EMPLOYE, un type d'entités *innocent*.

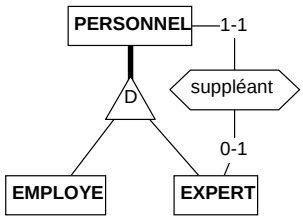


Figure A18.6 - Problème collatéral pour le type d'entités EMPLOYE

Ce schéma est assez facile à analyser (μ_{sEX} est la cardinalité moyenne du rôle suppléant.EXPERT) :

$$\mu_{sEX} \leq \mathbf{1}$$

$$\mathbf{N}_{PERSONNEL} = \mathbf{N}_{EXPERT} \times \mu_{sEX} \leq \mathbf{N}_{EXPERT}$$

$$\mathbf{N}_{\text{EXPERT}} \leq \mathbf{N}_{\text{PERSONNEL}}$$

PERSONNEL et EXPERT ont donc les mêmes populations (l'une est une partie de l'autre et elles sont de même taille). On en conclut que EMPLOYE n'est pas satisfiable et qu'il sera toujours vide. Conséquence de cette observation : la cardinalité [0-1] est en réalité [1-1]. En l'absence de contrainte de disjonction, EMPLOYE serait toujours une partie de EXPERT et en serait donc un sous-type !

Mentionnons une variante faible de la non-satisfiabilité : la ***non-initialisabilité***. La figure A18.7 reprend une structure que nous avons souvent utilisée, à ceci près que nous imposons ici que *toute personne ait un supérieur* (cardinalité [1-1]). Si à ce schéma doit, à tout instant, correspondre un graphe des instances acyclique (une personne ne peut être son propre responsable, ni directement ni indirectement), alors le type PERSONNE est *non initialisable*. Il est en effet impossible d'introduire la première entité PERSONNE puisque cette opération suppose que l'entité de son supérieur (qui lui est différente) soit déjà présente. En revanche, PERSONNE admet des instances valides et est donc satisfiable, même si on peut se demander comment la première a été créée[5] ! En revanche, si on relâche la contrainte d'acyclicité, et si on admet, au minimum, que certaines personnes puissent être leur propre supérieur, alors PERSONNE devient satisfiable et initialisable. On trouvera en fin de cette annexe, à titre d'exercices, d'autres schémas présentant les mêmes anomalies d'initialisabilité.

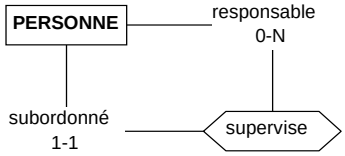


Figure A18.7 - Structure hiérarchique problématique

Les problèmes de non-satisfiabilité dus aux cardinalités ont été assez bien étudiés dans la littérature. On citera en particulier [Thalheim, 1992] et [Boufares, 2004].

Correction

Il est rare que des objets non satisfiables soient introduits délibérément dans un schéma. Il est donc indispensable de les repérer et de les corriger. Leur identification se base sur le fait que les problèmes surgissent lorsque, dans un circuit de types d'associations, un type d'entités est obligatoirement associé à lui-même. Le calcul

5. La reconnaissance de cette *anomalie utile* et sa résolution sont très anciennes, puisqu'on les trouve dans les premiers SGBD CODASYL (début des années 70 mais toujours utilisés aujourd'hui), sous la forme des types d'associations (set types) dont un rôle est mandatory manual. On en trouvera une description sur le site de l'ouvrage. L'adaptation de cette solution simple et élégante au mécanisme des clés étrangères aurait été très facile mais SQL ne l'a malheureusement pas adoptée, forçant le programmeur à recourir à des techniques complexes basées sur des transactions, des méthodes de tables typées ou des procédures SQL, qui seront décrites au chapitre 20.

des tailles des populations permettra de vérifier l'absence ou la présence de problèmes. On résoudra le problème en élargissant une des cardinalités ou en supprimant une contrainte d'acyclicité.

d) Contraintes d'inclusion incohérentes

La section 3.5 a montré que les contraintes d'inclusion avaient la propriété de transmettre d'une relation vers une autre des contraintes telles que les dépendances fonctionnelles et certaines contraintes d'inclusion. Ces contraintes *héritées* peuvent entrer en conflit avec des contraintes locales.

La figure A18.8 illustre deux situations typiques. Dans le schéma de gauche, la projection de assignée sur les rôles FOURNISSEUR et PIECE est clairement *plusieurs-à-plusieurs*. L'ensemble de couples qui lui correspond ne peut en toute généralité être une partie de la population de offre, qui est *un-à-plusieurs*. De même, le schéma de droite, dont la contrainte d'inclusion exprime le fait que *le directeur d'un service est un de ses employés*, posera un problème lorsqu'il s'agira de préciser le directeur d'un service qui n'a pas d'employé[6].

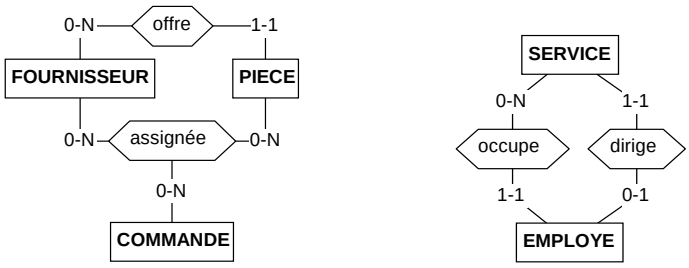


Figure A18.8 - Deux structures incohérentes

A18.2 PROCESSUS DE NORMALISATION DU SCHÉMA CONCEPTUEL

<complément de la section 18.7>

Un schéma correct doit encore satisfaire des critères de forme qui lui conféreront des qualités essentielles telles que la facilité d'utilisation des (futures) données, la lisibilité, l'expressivité, la concision, la clarté ou l'évolutivité. On dira d'une construction qui respecte ces critères qu'elle est **normalisée**. S'assurer qu'un schéma respecte ces critères et le corriger si tel n'est pas le cas est un processus bien identifié appelé **normalisation**.

Une construction non normalisée n'est pas erronée, au sens défini dans la section A18.1, mais elle ne correspond cependant pas aux *bonnes pratiques* recommandées

6. Le lecteur attentif se souviendra peut-être que ce cas a déjà été rencontré sous une autre forme à la section 12.7, comme illustration de constructions contradictoires.

par la profession. Elle est, dans un certain sens, *non optimale* ou *maladroite*. On mettra ici en cause, non sa sémantique, mais sa forme ou son style[7]. Normaliser une construction consistera à améliorer cette forme sans modifier ce qu'elle exprime.

Nous avons déjà rencontré le concept de *normalisation* au chapitre 3. Il s'agissait alors de transformer une relation de manière que chaque fait y soit représenté une et une seule fois. Une relation non normalisée n'est pas sémantiquement incorrecte, mais elle pose des problèmes de *lisibilité* (deux types de faits dans une même structure), de *gestion* des données (mises à jour complexes), de *place* occupée (redondance) et d'*évolutivité* (il est délicat d'introduire la représentation de nouveaux types de faits).

Un schéma conceptuel est sujet à des difficultés du même type, qui seront résolues également par des transformations qui amélioreront les qualités recherchées que nous venons de mentionner.

On admet que la principale préoccupation est la **lisibilité**, parfois appelée **compréhensibilité**. Cette qualité, qui n'est pas aisée à définir avec précision, mesure l'efficacité avec laquelle le schéma transmet l'intention de son auteur (la sémantique externe du schéma) à un lecteur quelconque, de compétence moyenne. Cette qualité est en effet essentielle dans tous les processus qui se basent sur l'exploitation d'un schéma de base de données : programmation, diffusion des données, administration des données, maintenance de la base de données, évolution de la base de données, etc. Comment en effet utiliser et manipuler de manière judicieuse des données dont on ne comprend pas parfaitement la signification et les contraintes qui leur sont imposées ?

On admet aussi que la qualité de lisibilité est favorisée par un nombre limité de caractéristiques formelles (donc identifiables avec une certaine précision, voire mesurables). Nous en retiendrons sept, que nous allons détailler dans cette section :

- **simplicité** (ou **minimalité**) : un type de faits doit être représenté le plus simplement possible ; il est inutile par exemple de représenter une simple propriété par un type d'entités (A18.2.1) ;
- **expressivité** : un type de faits doit être représenté par une construction qui évoque naturellement et clairement sa nature ; c'est ainsi par exemple qu'on utilisera une relation *is-a* pour indiquer qu'un type d'entités est un cas particulier d'un autre (A18.2.2) ;
- **absence de sur-spécifications** : le schéma représente les types de faits raisonnablement utiles, sans préjuger de la manière dont ils seront implémentés (A18.2.3) ;
- **absence de constructions anormales** : on évitera notamment les constructions *anormales* (cas limites d'utilisation d'une construction, style inhabituel) ou *irrégulières* (changement injustifié de style, style incohérent); (A18.2.4) ;

7. On appellera *style* d'un schéma l'ensemble des raisonnements systématiques qui président au choix des règles de représentation des types de faits. Un style est propre à un concepteur (on reconnaît un auteur à son style), à une méthode ou à une école de pensée.

- **régularité d'expression** ou **prévisibilité** : le lecteur ne doit pas être *surpris* par l'usage d'une construction pour exprimer un certain type de faits ; on évitera notamment les changement injustifié de style et les styles incohérents; (A18.2.5) ;
- **absence de redondance** : un schéma doit exprimer un type de faits une et une seule fois ; il en est de même des instances futures ou existantes des constructions ; on reconnaîtra ici une qualité similaire à celle qui est recommandée pour les schémas relationnels au chapitre A18.3 (A18.2.6, A18.2.7 et A18.2.8) ;
- **qualité graphique** : la manière dont les constructions d'un schéma sont disposées dans l'espace graphique peut améliorer considérablement sa lisibilité ; il n'est pas recommandé par exemple de recouvrir le type d'entité DETAIL par le type d'entités COMMANDE (de sorte qu'il soit invisible) ou au contraire de le positionner 10 mètres à droite (A18.2.9) ;
- **respect des standards** : le schéma doit respecter certaines règles imposées par l'organisme pour lequel (ou dans lequel) il est développé, quand bien même le concepteur les estimerait inutiles, voire absurdes (A18.2.10).

La normalisation est un processus riche et complexe. Il est essentiel dans toute méthode de conception d'une base de données, en particulier lorsque le schéma conceptuel est construit en plusieurs étapes par des concepteurs différents, à différentes époques. Il s'avérera aussi particulièrement utile lorsque les schémas sont produits de manière semi-automatique, notamment lors de l'extraction à partir de formulaires, d'écrans ou lors de l'analyse automatique de texte. Il constituera enfin un processus essentiel en rétro-ingénierie, lorsque les schémas extraits sont fortement *colorés* (ou *pollués*) par les caractéristiques des SGBD sources ou par la maladresse des générations de concepteurs qui sont intervenus sur ces schémas au cours du temps.

A18.2.1 Simplification des constructions non minimales

Une construction non minimale traduit un type de faits pour lequel il existe une représentation plus légère généralement jugée plus appropriée. Un trop grand nombre de ces constructions risque de rendre la lecture du schéma plus laborieuse et donc d'affecter négativement sa maintenabilité. C'est à l'analyste de juger de l'opportunité de corriger ce problème. Attention cependant, un excès de correction peut conduire à un défaut d'expressivité ou peut contrarier l'évolutivité du schéma. En effet, une construction excessive peut n'être que temporaire. Par exemple, un type d'entités attribut pourrait recevoir d'autres attributs dans un avenir proche.

a) Type d'entités attribut

<voir ouvrage, 3e édition>

b) Type d'associations n-aire avec rôle [1-1] ou [0-1]

<voir ouvrage, 3e édition>

c) Attribut composé d'un seul composant

<voir ouvrage, 3e édition>

d) Type d'associations un-à-un à rôles obligatoires

<voir ouvrage, 3e édition>

e) Sous-type faiblement spécifique

<voir ouvrage, 3e édition>

f) Type d'entités doté d'un seul sous-type sans attributs ni rôle

<voir ouvrage, 3e édition>

g) Contraintes d'existence ternaires sous forme binaire

Il est possible d'exprimer certaines contraintes d'existence portant sur n composants sous la forme d'une conjonction de contraintes portant sur n-1 composants, expression complexe à laquelle on préférera la forme n-aire, plus expressive.

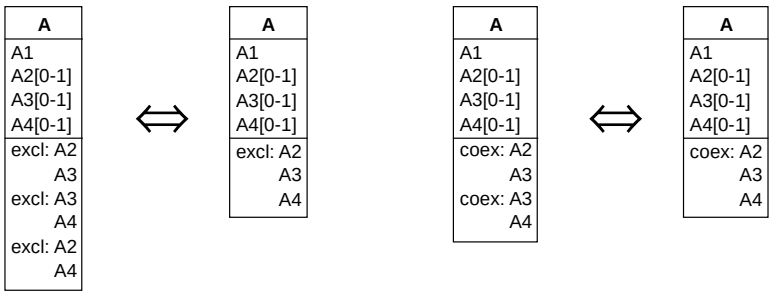


Figure A18.9 - Simplification de contraintes d'existence

On a, en particulier :

(excl: A,B,C) = (excl: A,B) $\wedge$ (excl: B,C) $\wedge$ (excl: A,C)

(coex: A,B,C) = (coex: A,B) $\wedge$ (coex: B,C)

La figure A18.9 reprend ces deux relations sous forme de schémas équivalents. Le schéma de droite, plus simple, sera toujours préféré.

A18.2.2 Explicitation des constructions insuffisamment expressives

Une construction *insuffisamment expressive* traduit un type de faits pour lequel il existe une représentation plus appropriée, qui évoque naturellement et clairement la nature de celui-ci. Elle trouve souvent son origine dans l'analyse de formulaires dont la structure est trop pauvre (listes, arbres) pour représenter explicitement les relations entre les données. Une telle construction n'est pas une erreur à proprement parler, mais elle risque d'induire des difficultés de compréhension du schéma, et donc d'affecter négativement sa maintenabilité. Il est conseillé de la corriger.

a) Type d'entités association

<voir ouvrage, 3e édition>

b) Attribut complexe

<voir ouvrage, 3e édition>

c) Attribut de référence

<voir ouvrage, 3e édition>

d) Attributs sériels et concaténés

<voir ouvrage, 3e édition>

e) Contrainte de coexistence (coex)

<voir ouvrage, 3e édition>

f) Contrainte d'implication (if)

<voir ouvrage, 3e édition>

g) Identifiant semi-obligatoire

<voir ouvrage, 3e édition>

h) Relations is-a implicites

<voir ouvrage, 3e édition>

A18.2.3 Réduction des sur-spécifications

Sur-spécifier c'est, dans le contexte qui est le nôtre, *en dire trop*. Une construction constitue une sur-spécification si son implémentation ultérieure imposera des contraintes trop fortes, soit à l'utilisateur des données, soit à l'implémenteur, soit encore aux autres composants du système d'information. Cet excès peut prendre différentes formes. Nous en illustrerons deux parmi les plus fréquentes.

a) Intégrité et plausibilité

Il est utile de distinguer une *contrainte d'intégrité* d'une *propriété de plausibilité*. La première est impérative et ne souffre aucune exception : toute tentative de violation doit être rejetée. La seconde décrit une situation à ce point habituelle que tout écart par rapport à celle-ci devrait attirer l'attention. Toute tentative de violation ne doit pas nécessairement être rejetée, mais devrait donner lieu à un avertissement. Par exemple, il est anormal que deux clients distincts ayant mêmes noms, prénoms et dates de naissance soient en plus domiciliés à la même adresse. Si, par extraordinaire, un tel cas devait se produire, alors il serait inopportun de le refuser, mais il serait prudent d'en avertir le responsable des données afin qu'il vérifie. Il serait en tout cas maladroit de traduire cette règle par un identifiant (figure A18.10).

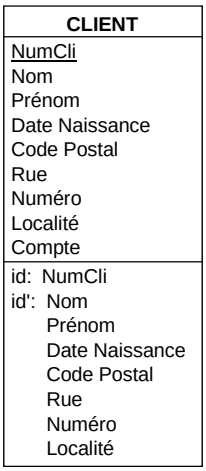


Figure A18.10 - Schéma sur-spécifié : l'identifiant secondaire n'est qu'une simple propriété de plausibilité qui ne mérite pas d'être déclarée comme un identifiant

b) Relaxation de contraintes trop fortes

La tentation est grande, lorsqu'on fixe les contraintes, de considérer la base de données comme fonctionnant en *régime normal*. C'est oublier qu'elle évolue et qu'elle passe par des phases où son état est incomplet tout en étant valide. Il importe aussi d'envisager la manière dont l'information sera gérée. Des questions d'ergonomie ou de réglementation propres à l'organisation peuvent imposer des états a priori imprévu des données. Considérons l'exemple des commandes, dont on sait qu'elles référencent chacune au moins un produit. Toute commande doit donc posséder au moins un détail (figure A18.11, gauche).

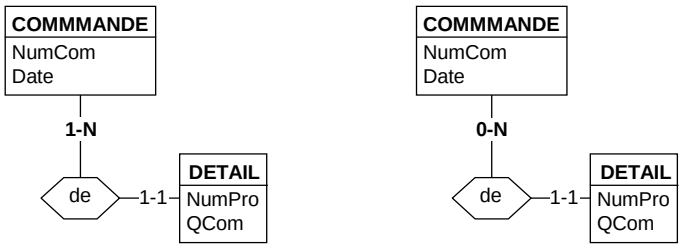


Figure A18.11 - Relaxation d'une contrainte pour tenir compte du comportement organisationnel

Ce raisonnement ignore deux situations liées au fonctionnement de l'entreprise. La première est que les bons de commandes sont encodés par un premier service qui enregistre l'en-tête et vérifie l'existence du client. Plus tard, un second service encode les détails des commandes préenregistrées. La seconde situation, rare mais pas impossible, est celle d'une commande dont tous les produits viennent d'être

retirés des stocks. Le concept d'une commande sans détail, a priori absurde, ne doit donc pas être interdit (figure A18.11, droite).

A18.2.4 Traitement des constructions anormales

Une construction est dite *anormale* lorsqu'elle n'est pas conforme à ce qui se pratique habituellement et apparaît donc comme une *idiosyncrasie*. Certaines constructions non minimales peuvent être jugées anormales. Ces constructions se retrouveront dans des schémas développés par des analystes peu expérimentés, mais plus souvent dans des schémas construits de manière automatique, par exemple en rétro-ingénierie ou par des interfaces objet-relationnel (*Hibernate* par exemple). Une construction anormale ne constitue pas réellement une erreur, mais elle diminue la lisibilité du schéma et donc sa maintenabilité. Il est donc conseillé de la modifier.

a) Constructions dégénérées

Une construction est qualifiée de *dégénérée* lorsque sa composition est considérée comme *limite*, au point que son usage n'apparaît plus comme justifié. Par exemple (figure A18.12), déclarer pour le type d'entités CLIENT une contrainte de disjonction (D) ou de totalité (T) suppose qu'il y ait au moins deux sous-types. Une contrainte d'existence porte normalement sur un groupe d'au moins deux composants, un attribut composé comprend normalement au moins deux composants. Les schémas de la figure A18.12 ne comportent pas véritablement d'erreurs, mais plutôt des constructions inutiles.

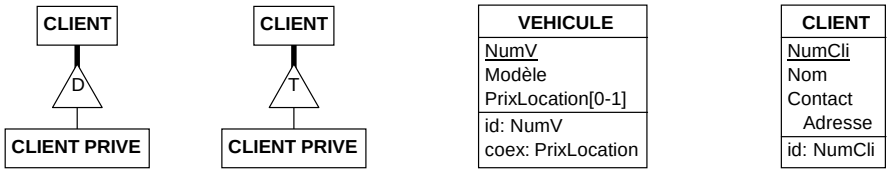


Figure A18.12 - Quatre constructions dégénérées

Notons cependant que le deuxième schéma est un peu particulier : il implique que les populations de CLIENT et de CLIENT PRIVE sont identiques. Si tel est l'intention de l'analyste, peut-être vaudrait-il mieux n'en conserver qu'un seul.

b) Type d'entités sans attributs, sans rôles, sans identifiants

Normalement, un type d'entités comporte des attributs, joue des rôles dans des types d'associations et possède au moins un identifiant. Il est utile de vérifier la validité d'un type d'entités qui ne respecterait pas ces propriétés.

c) Constructions étrangères

Cette catégorie recouvre une grande variété de situations qui ont toutes une origine commune : l'intrusion dans un schéma d'une construction appartenant à un autre

niveau d'abstraction ou à un autre modèle[8]. Ce phénomène est particulièrement perceptible dans deux contextes.

- *Migration de bases de données*. Le schéma décrit une base de données qui résulte de la migration rapide d'une base de données ancienne vers une technologie plus moderne. C'est ainsi qu'on retrouve dans une base de données SQL des constructions propres aux modèles CODASYL, IMS ou TOTAL/IMAGE dans lesquels la base de données ancienne était implémentée. Le schéma conceptuel actuel de cette base de données conserve des constructions visiblement héritées de ces modèles.
- *Culture étrangère*. L'analyste est fortement imprégné d'une culture de modélisation étrangère à celle qui s'impose dans la construction du schéma en cours. Le programmeur Java expérimenté aura tendance à raisonner en termes de classes, et associera à chaque table un *object-id*, c'est-à-dire un identifiant technique. Le concepteur IMS ou l'expert XML auront tendance à construire des schémas hiérarchiques.

Le schéma de la figure A18.13, en principe équivalent à celui de la figure 16.18, attributs exclus, pourrait avoir été construit par un expert IMS[9]. Si le métissage culturel est source de richesse dans notre société, il est en revanche déconseillé dans le domaine de la modélisation des données, du moins dans un même schéma. Les constructions étrangères diminuent la lisibilité du schéma et peuvent rendre problématiques sa maintenabilité et son évolutivité.

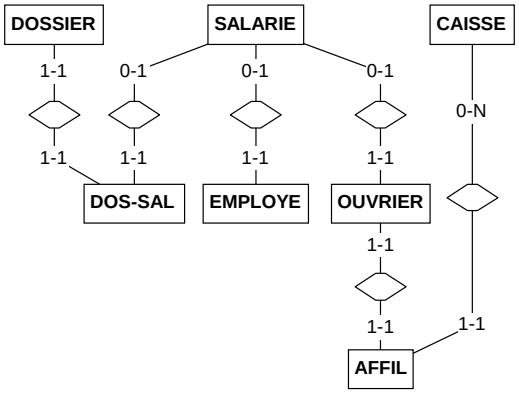


Figure A18.13 - Un schéma conceptuel fortement coloré *IMS*

Le schéma de la figure A18.14 comporte de (trop) nombreuses hiérarchies *is-a* (outre un type d'entités non satisfiable). Il est probable que son concepteur, habitué

8. On dira aussi issue d'un autre *paradigme*.
9. En IMS, DOS-SAL aurait été un *logical child* de DOSSIER et AFFIL un *logical child* de CAISSE [Hainaut, 2009b].

de la programmation orientée objet, a bien pris soin de définir une classe pour chaque collection d'objets dotée d'au moins une propriété spécifique (un attribut que d'autres objets ne possèdent pas). Un schéma conçu par un expert en modélisation conceptuelle contient généralement un petit nombre de hiérarchies peu profondes[10].

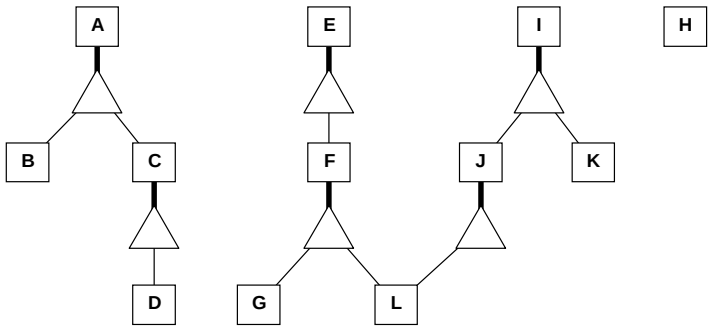


Figure A18.14 - Un schéma *un peu trop proche* de la programmation OO[11]

A18.2.5 Uniformisation des constructions irrégulières

Un schéma comporte des constructions irrégulières lorsque des types de faits similaires y sont traduits par des constructions de nature différente. Les différents phénomènes décrits dans ce chapitre montrent qu'il existe souvent plusieurs manières de représenter des types de faits tels que les suivants :

- *des associations* : représentées par un type d'associations bien sûr, ou des types d'entités association, mais parfois par de simples attributs de référence ; cette dernière structure est très fréquente pour les types d'entités de dimension (annexe 16), tels que ANNEE ou DEPARTEMENT, auxquels sont associés un grand nombre d'autres types d'entités ; dans ce cas, le concepteur associera la dimension tantôt via un type d'associations tantôt, lorsqu'il estime que le schéma devient *encombré*, par un attribut de référence ;
- *une propriété multivaluée* : par un attribut multivalué, mais aussi par un type d'entités représentant les valeurs ou un type d'entités représentant les instances ;
- *des associations de plus de deux entités* : type d'associations n-aire, type d'entités association, attribut multivalué ;
- *composition d'entités* : type d'association simple, type d'associations de composition, attribut multivalué.

Les irrégularités syntaxiques se marqueront dans le choix des noms des objets du schéma. Un schéma dans lequel on trouve les types d'entités dénommées CLIENT,

10. On y observe notamment que moins d'un quart des types d'entités participent à une hiérarchie *is-a*.
11. Ce n'est pas son seul défaut ! Quel est l'autre ?

Commande et PRODUITS présente des irrégularités à corriger : *majuscule* ou *minuscules*, *singulier* ou *pluriel*.

Cette situation est souvent due à l'intervention de plusieurs concepteurs de culture et de formation différentes. Dans des schémas anciens, les mises à jour successives, parfois réalisées dans l'urgence, introduisent des irrégularités. Il est vivement conseillé de régulariser ces constructions dans un schéma.

A18.2.6 Élimination des redondances de contrainte

Une construction d'un schéma (le plus souvent une contrainte) est dite redondante lorsqu'elle peut se déduire des autres propriétés du schéma. Il ne lui correspond pas de redondance d'instances. Il est conseillé de la supprimer.

a) Relations is-a transitives

Si D est un sous-type de B et si ce dernier est un sous-type de A, alors D est aussi un sous-type de A (figure A18.15, schéma de gauche). On dira que la relation *is-a* est *transitive*. La justification est assez évidente tant du point de vue de la propriété d'inclusion des populations ($D \subseteq B \subseteq A$) que du principe de l'héritage (D hérite de B qui hérite de A). On n'indiquera pas les relations *is-a* transitives (schéma central).

Le lecteur curieux analysera avec attention le schéma de droite de la même figure. Ce schéma comporte non seulement une contrainte redondante, mais la contrainte de disjonction introduit une construction erronée. La population de D est un sous-ensemble de celle de B tout en en étant disjointe (contrainte **D**). Elle est donc vide par construction, de sorte que D est non satisfiable.

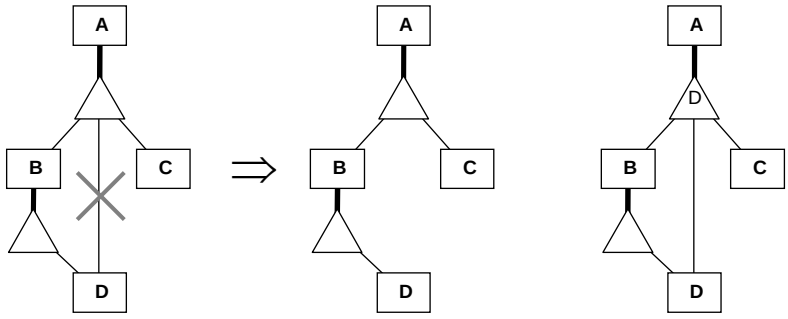


Figure A18.15 - Relation is-a transitive et type d'entités non satisfiable (à droite)

b) Identifiants redondants

Les identifiants implicites d'un type d'entités se déduisent des propriétés des types d'associations auxquels celui-ci participe (section 16.10.1).

De même, les identifiants implicites d'un type d'associations se déduisent des propriétés de ses rôles (section 16.10.2). Les valeurs d'un attribut multivalué du type *ensemble* sont par construction distinctes. Il est donc inutile de spécifier ces identifiants de manière explicite (figure A18.16).

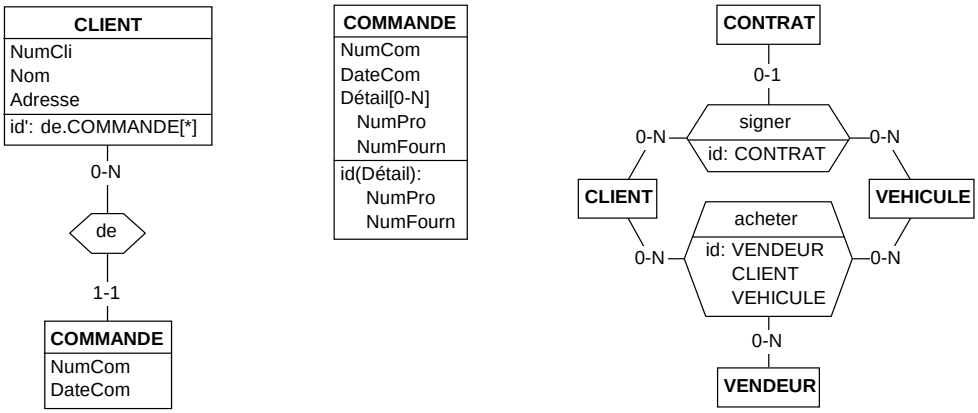


Figure A18.16 - Les quatre identifiants de ces schémas sont redondants et devraient être supprimés

c) Identifiants non minimaux

Un groupe de composants est un identifiant non minimal lorsqu'il est possible de lui retirer un composant sans lui faire perdre son statut d'identifiant. Sauf pour des raisons à justifier explicitement[12], un identifiant dont on peut déduire qu'il est non minimal à partir des propriétés des objets du schéma doit être supprimé. Les identifiants {Professeur, Matière} et {de.REGION, Nom} de la figure A18.17 peuvent être supprimés sans perte d'information. Le caractère non minimal peut également être dû à la présence d'une dépendance fonctionnelle au sein des composants de l'identifiant. Les raisonnements de détection et de correction ont été décrits au chapitre 3.

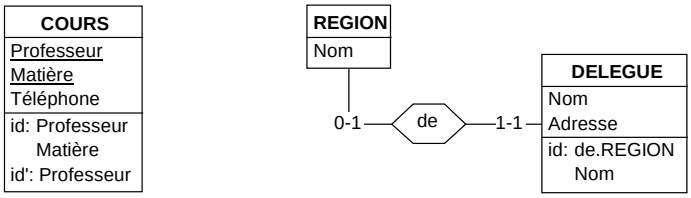


Figure A18.17 - Les identifiants non minimaux doivent être supprimés

d) Contraintes d'existence dérivées

Une contrainte d'existence qui est déductible des autres contraintes est redondante et doit donc être supprimée. Dans l'exemple de la figure A18.18 (gauche), on peut montrer que la deuxième ou la troisième contrainte est redondante. On a en effet, pour trois composants arbitraires A, B, C :

12. Voir la section A18.3.8.5 par exemple, de laquelle nous avons repris le premier schéma.

(coex: A,B) ∧ (excl: A,C) = (coex: A,B) ∧ (excl: B,C)

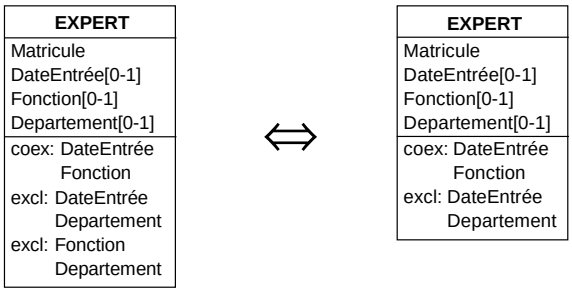


Figure A18.18 - Une des contraintes d'exclusion est inutile

A18.2.7 Élimination des redondances structurelles d'instances

<voir ouvrage, 3e édition>

a) Type d'associations et clé étrangère

<voir ouvrage, 3e édition>

b) Attributs copiés et attributs calculables

<voir ouvrage, 3e édition>

c) Types d'associations composés et projetés

<voir ouvrage, 3e édition>

d) Types d'entités association et types d'associations

<voir ouvrage, 3e édition>

e) Constructions en double

<voir ouvrage, 3e édition>

A18.2.8 Élimination des redondances internes

<voir ouvrage, 3e édition>

a) Normalisation d'un type d'entités

<voir ouvrage, 3e édition>

b) Normalisation d'un type d'associations

<voir ouvrage, 3e édition>

A18.2.9 Amélioration de la présentation graphique

Un schéma correct et normalisé selon les critères étudiés peut être proprement illisible. Les figures 16.42 et A18.19 représentent le même schéma, mais il est patent que ce dernier est pratiquement illisible, voire inutilisable pour un lecteur humain. En quoi ces deux dispositions se distinguent-elles ? La réponse n'est pas simple et relève tout à la fois de caractéristiques psychologiques (cognitives), culturelles, voire sociologiques.

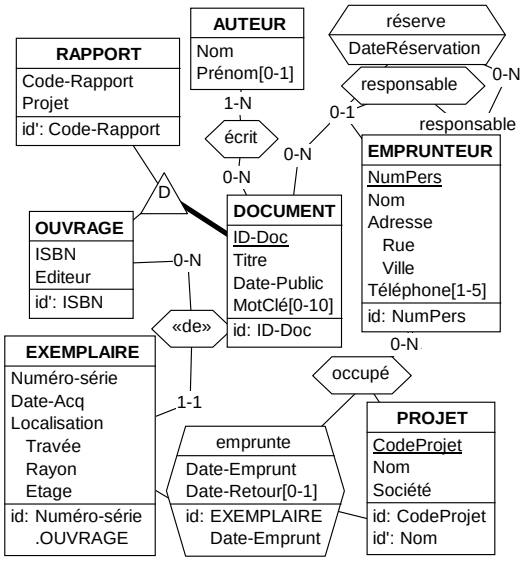


Figure A18.19 - Un schéma correct et normalisé mais illisible

Tout comme pour un texte, la forme doit évoquer visuellement de la manière la plus claire possible la structure et le contenu. Il est bien connu que les titres, la forme des paragraphes, la numérotation, les polices et les styles utilisés contribuent à la compréhension de la structure d'un exposé et en facilitent la lecture. La disposition graphique est donc un langage qui exprime une partie du contenu du schéma. Par exemple, deux objets sémantiquement proches devraient être graphiquement proches l'un de l'autre.

La qualité visuelle d'un schéma est d'une grande importance puisqu'elle conditionne sa lisibilité, et par là, la facilité et la fiabilité des modifications ultérieures. Quelles sont donc les règles de positionnement que nous devrions adopter pour dessiner au mieux nos schémas Entité-association ? Nous essayerons d'en identifier les principales, que nous classerons en trois catégories de spécificité croissante.

a) Les règles de disposition de graphes

La disposition visuelle d'un graphe quelconque, constitué de nœuds et d'arcs, doit obéir à quelques règles évidentes que dicte le simple bon sens. Ces règles s'appliquent également à tout schéma Entité-association.

Dans celui-ci, on considérera que les types d'entités, les types d'associations, les rôles et les ensembles de sous-types (triangles) sont des *nœuds* et que les traits qui relient ces objets dans le schéma sont des *arcs*.

1. Deux nœuds sont suffisamment écartés pour qu'on puisse clairement les distinguer. On évitera par exemple de superposer deux types d'entités ou de cacher un rôle derrière un type d'associations.
2. Un nœud est suffisamment éloigné de tout arc auquel il n'appartient pas. Dans le cas contraire, on risquerait de considérer à tort ce nœud comme une extrémité de cet arc. Ainsi, on ne positionnera pas un rôle sur un arc d'un autre type d'associations.
3. Un arc est aussi court que possible. Un arc relie deux nœuds entre lesquels existe une relation logique. On cherchera donc à les rapprocher visuellement.
4. Les croisements d'arcs doivent être réduits au minimum. Suivre un arc pour repérer le nœud à son extrémité est plus difficile lorsqu'il faut ignorer les autres arcs qui le croisent.
5. Si deux arcs doivent se croiser, leur angle doit être important. Si l'angle est trop petit, il existe un risque de confusion des deux arcs.

b) Les règles de disposition de schémas Entité-association

Le modèle Entité-association est conçu pour modéliser des situations bien spécifiques : les catégories d'objets statiques constituant le domaine d'application et leurs propriétés générales. Dans la vie courante, il existe déjà des représentations graphiques de ces objets, qui dépendent des relations qui unissent ces derniers. Par exemple, si deux catégories sont en relation hiérarchique, on dessinera dans un organigramme d'entreprise l'une au-dessus de l'autre pour suggérer que la première est hiérarchiquement supérieure à la seconde ou qu'elle l'englobe. Il est logique d'adopter des règles similaires dans les schémas conceptuels. Nous reprendrons quelques règles importantes :

1. *Structures hiérarchiques*. Un type d'associations 1:N représente souvent une structure hiérarchique, dans laquelle on distingue un type d'entités supérieur et un type d'entités subordonné, comme l'illustre la figure 12.25. On placera de préférence le premier au dessus du second pour symboliser cette relation.
2. *Structures de composition et de matérialisation*. Les types d'associations génériques étudiés à la section 16.14 sont définis entre un type d'entités majeur (composé, abstraction) et un type d'entités mineur (composant, matérialisation). Il est logique de positionner le premier au-dessus du second, comme l'illustre la figure 16.42
3. *Structures de spécialisation*. Selon un raisonnement similaire, on placera un sous-type sous son surtype. À défaut, on cherchera une représentation symétrique : disposition des sous-types en étoile autour du surtype ou sous-types alignés sur le côté ou au-dessus du surtype.

4. *Structures n-aires*. Pour chaque type d'associations n-aire, on cherchera à réaliser une certaine symétrie entre les types d'entités d'une part et le type d'association d'autre part : disposition en étoile ou types d'entités alignés.
5. *Structures non hiérarchiques*. Lorsqu'un type d'associations binaire n'induit aucune relation hiérarchique ou de dépendance, on placera autant que possible les types d'entités impliqués au même niveau, pour suggérer une certaine *collégialité*. On vise ici surtout les types d'associations 1:1 et N:N dont les deux rôles sont facultatifs.
6. *Ordre des éléments dans les groupes*. Les attributs d'un identifiant primaire seront idéalement placés en tête de la liste des attributs, de manière à suggérer l'importance du rôle qu'ils jouent. Dans le groupe des composants d'une contrainte, on placera les rôles en premier puisqu'un type d'entités est plus important qu'un attribut.

c) Les règles de disposition liées au domaine d'application

Ces règles dépendent du domaine d'application modélisé et ne peuvent être formulées en toute généralité. Elles régissent le regroupement de types d'entités et la disposition relative des groupes ainsi formés.

Une commande concerne un tiers de l'entreprise (client ou fournisseur) et référence à un article (à envoyer ou à commander), ce que représente la figure A18.20. On pourrait a priori penser à une disposition graphique symétrique des deux types de commandes. Or, il n'en est rien : (1) le client est considéré comme plus important que l'article, de sorte que COMMANDE-CLIENT est positionné près de CLIENT et (2) l'article est considéré comme plus important que le fournisseur, ce qui se traduit par la proximité de COMMANDE-FOURNISSEUR et d'ARTICLE.

De manière plus générale, les types d'entités relatifs à un sous-domaine (service du personnel, marketing, gestion des emprunteurs) seront souvent regroupés de manière à former les sous-schémas à forte cohésion et topologiquement identifiables.

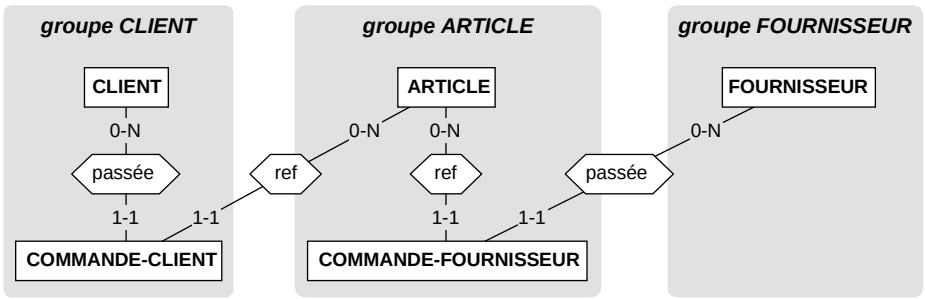


Figure A18.20 - Dissymétrie dans la perception de deux types de commandes

Quatre remarques

- La qualité graphique d'un schéma est favorisée par la symétrie et la régularité. On disposera des ensembles d'objets similaires de manière similaire.

- En toute généralité, l'ensemble de ces règles n'est pas cohérent. Certaines règles en effet sont contradictoires. Par exemple, tenter de raccourcir les arcs entraîne le plus souvent l'augmentation des croisements d'arcs.
- Certaines contraintes pratiques peuvent conduire à violer les règles de positionnement énoncées. Tel est le cas de la minimisation de l'espace occupé (le schéma doit *tenir* sur une page A4).
- Ces règles sont bien entendu d'application pour les diagrammes de classes UML. On observe cependant un certain laxisme dans de nombreux schémas publiés dans la littérature, où la contrainte de minimisation de l'espace prévaut souvent.

A18.2.10Mise en conformité aux standards en vigueur

Un schéma jugé satisfaisant dans l'absolu peut violer des standards, généralement restrictifs, que l'organisation s'est imposés pour ses propres développements. Le modèle décrit au chapitre 16 et utilisé dans le présent chapitre étant relativement riche, un tel standard constitue ce qu'on appellera un *sous-modèle*. Par exemple, le modèle de base étudié au chapitre 12 exclut notamment les relations *is-a*, les attributs multivalués, les attributs composés, les types d'associations n-aires, les attributs de types d'associations et les contraintes d'existence. Il constitue un sous-modèle spécifique simple fréquemment adopté.

Le concepteur devra s'approprier ces règles et s'y conformer lors de l'élaboration de schémas conceptuels. Nous décrirons brièvement quelques règles limitant les constructions acceptées dans un schéma conceptuel ainsi que des transformations de mise en conformité.

a) Limitations concernant les hiérarchies is-a

Seules certaines des quatre configurations décrites à la figure 16.14 peuvent être utilisées. C'est ainsi que les sous-types non disjoints et/ou non couvrants pourront être interdits. Nous limiterons la discussion au cas de deux sous-types.

Lorsque les sous-types B et C doivent être **disjoints**[13] (contrainte **D**) on représentera explicitement leur intersection par un sous-type BC supplémentaire (figure A18.21). Les sous-types d'origine disparaissent au profit des types d'entités B' et C' ne représentant plus que les entités appartenant exclusivement à l'un d'entre eux. On obtient donc trois sous-types disjoints B', C' et BC, dont les populations sont définies comme suit (selon la notation de la section 15.4) : $B' = B - C$; $C' = C - B$; $BC = B \cap C$. Cette transformation fait l'hypothèse que les sous-types B et C soient terminaux (eux-mêmes n'ont pas de sous-type). En effet, si l'un d'eux possède un sous-type, il est très difficile de l'éclater en types disjoints[14]. On trouvera une discussion plus détaillée du traitement des sous-types non disjoints à l'annexe G.

13. Cette limitation à des sous-types disjoints rapproche le schéma conceptuel des hiérarchies de classes Java et de TDU des SGBD relationnels objet.
14. Cependant, s'il existe un type d'entité D dépendant simultanément des sous-types d'origine B et C, on le rattache désormais au nouveau sous-type BC.

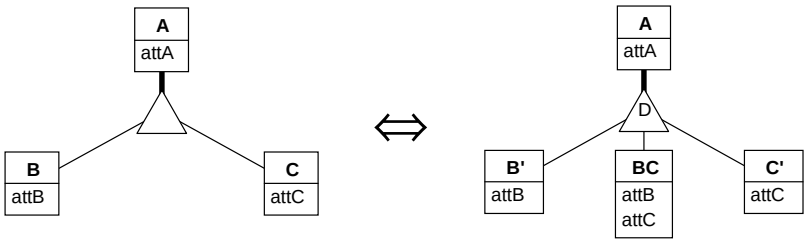


Figure A18.21 - Transformation symétrique d'une hiérarchie is-a quelconque en disjonction

Rendre disjoint un ensemble de plus de deux sous-types est beaucoup plus complexe. Par exemple, trois sous-types B, C et D non disjoints produisent les sept[15] sous-types disjoints suivants, dont on spécifie les populations : B' = B - C - D; C' = C - B - D; D' = D - B - C; BC = B∩C - D; BD = B∩D - C; CD =C∩D - B ; BCD = B∩C∩D.

L'exigence de sous-types **couvrants** (contrainte **T**) se traite par l'adjonction d'un sous-type dont la population rassemble toutes les entités du surtype qui n'appartiennent à aucun sous-type d'origine. Ce sous-type ne possède pas d'attributs propres et ne joue aucun rôle propre.

La combinaison des deux contraintes consiste à exiger que l'ensemble des sous-types forme une **partition** (figure A18.22 où A' = A - B - C).

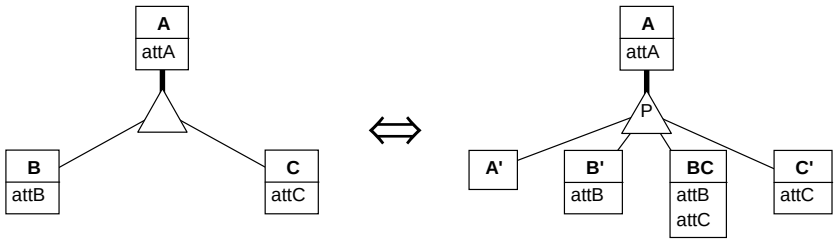


Figure A18.22 - Transformation d'une hiérarchie is-a quelconque en partition

b) Limitations concernant les attributs

Le sous-modèle local peut exclure les attributs multivalués, composés ou facultatifs. Les attributs **multivalués** seront transformés par représentation des valeurs ou des instances selon les deux techniques décrites à la figure 18.25 (lues de droite à gauche). Un attribut **composé** sera transformé de deux manières : par *désagrégation* ou par transformation en un *type d'entités* (figure A18.23). Dans ce dernier cas, on pourra appliquer, comme illustré à la figure 18.26, la représentation des valeurs ou des instances. Un attribut monovalué **facultatif** sera transformé en un type d'entités

15. La disjonction d'un ensemble de n sous-types produit $2^n - 1$ sous-types disjoints.

selon les transformations de la figure 18.26, le rôle joué par le type d'entités d'origine étant facultatif.

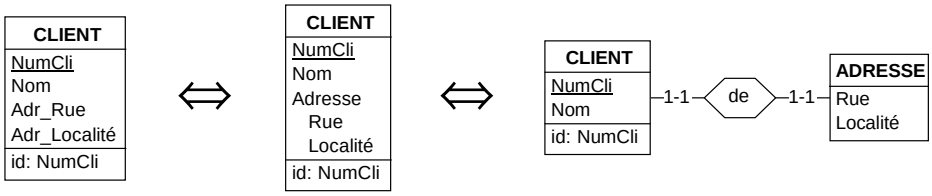


Figure A18.23 - Transformation d'un attribut composé (centre) par désagrégation (gauche) et par représentation des instances (droite)

c) Limitations concernant les types d'associations

L'interdiction des types d'associations n-aires, avec attributs, cycliques, voire *plusieurs-à-plusieurs* entraînera la transformation de ceux-ci en *type d'entités association*. Nous avons déjà rencontré cette technique à plusieurs reprises, notamment aux figures 18.32, 18.33 et 18.37 (schémas 2 et 4).

d) Limitations concernant les contraintes

Toutes les contraintes ne sont pas nécessairement recommandées. Pour ce qui concerne les **contraintes d'existence**, nous avons vu qu'il était possible de transformer certaines d'entre elles en structures (section 18.7.2, paragraphes e et f). Les autres pourront souvent être transformées en une hiérarchie *is-a* (section 18.7.2, paragraphe h).

La composition des **identifiants** peut également faire l'objet de limitations. Il n'est pas rare qu'on limite la composition des identifiants primaires à deux, voire à un seul composant. De même, les composants trop longs (Titre d'OUVRAGE) ou trop complexes (l'attribut composé Adresse) peuvent être exclus. Dans de tels cas, un identifiant violant cette contrainte sera transformé en identifiant secondaire et remplacé par un identifiant primaire technique, constitué, par exemple, d'un numéro abstrait sans signification (figure A18.24).

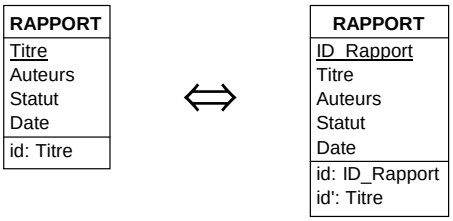


Figure A18.24 - Remplacement d'un identifiant primaire

e) Limitations syntaxiques et lexicales

L'organisation peut imposer des règles de formation des **noms** des objets dans un schéma. Citons-en quelques-unes à titre d'exemple et non de modèle :

- un nom ne peut comporter certains caractères (espace, signe de ponctuation, symbole mathématique, etc.); un nom ne peut dépasser **n** caractères ;
- un nom ne peut appartenir à une liste de noms réservés ;
- le nom d'un attribut est préfixé par l'abréviation de son type d'entités (CLI-ADRESSE, COM-DATECOM) ;
- un attribut identifiant est préfixé par les caractères « Id » (IdClient, IdProduit) ;
- le nom d'un type d'associations est un verbe à l'infinitif ;
- le nom d'un type d'entités est un substantif au singulier.

f) Limitations de modèle

Les limitations peuvent être définies par un sous-modèle public indépendant de l'organisation, propre par exemple à un AGL, à une SSCI ou à un ouvrage publié. Les modèles utilisés dans [Elmasri, 2006][16], [Connolly, 2006], [Garcia-Molina, 2008] ou [Akoka, 2001] en sont quelques exemples.

A18.3PROTOTYPAGE D'UNE BASE DE DONNÉES

<Cette section constitue un développement pratique de la section 18.8.2>

Une base de données étant essentiellement un *composant passif* du système d'information, il nous faut préciser ce qu'on entend par *prototype* d'une base de données, terme généralement consacré aux fonctions de traitement et aux interfaces utilisateur.

Étant donné un schéma conceptuel, ou une partie de celui-ci, on envisage de livrer aux futurs utilisateurs de la base de données en projet; d'une part, une base de données SQL opérationnelle, et, d'autre part, un gestionnaire interactif doté d'une interface conviviale permettant à ces utilisateurs de créer, manipuler et examiner un contenu représentatif de cette base de données. Il n'est pas nécessaire que cette base de données soit optimisée ni que le volume des données introduites soit important : un prototype n'est en effet qu'une version réduite et simplifiée du système final. Par exemple, il est inutile à ce stade de créer des index, de préciser la politique de gestion des tampons ou de définir le plan de contrôle des accès.

La production d'un schéma SQL qui exprime le schéma conceptuel est une opération en général assez simple, et donc automatisable pour autant qu'on ne soit pas trop regardant sur la gestion des contraintes d'intégrité complexes. Pour ce qui

16. A noter cependant que le modèle d'Elmasri inclut une construction non reprise dans celui de cet ouvrage : la *catégorie*, qui est un sous-type d'une union de deux ou plusieurs types d'entités.

nous concerne, l'utilisation de l'atelier DB-MAIN permettra de générer de manière automatique le code SQL-DDL de la base de données prototype selon le SGBD qui sera utilisé par le prototype (typiquement SQLite, conformément à la version de base de SQLfast).

Nous examinerons le concept de gestionnaire interactif prototype en deux temps.

Nous développerons d'abord *manuellement* un ensemble de fonctions permettant à l'utilisateur de *jouer* avec une base de données prototype représentative pour qu'il puisse identifier les éventuelles erreurs de spécifications. Nous choisirons la base de données CLICOM.db, que nous avons largement utilisée dans les premiers chapitres de cet ouvrage, Nous lui ajouterons, pour compliquer quelque peu le travail, une table FACTURE, référençant DETAIL, et munie en outre d'une clé étrangère cyclique (figure A18.25). Cette base de données reçoit le nom de CLICOM-proto.db. Le contenu de la table FACTURE est illustré à la figure A18.26.

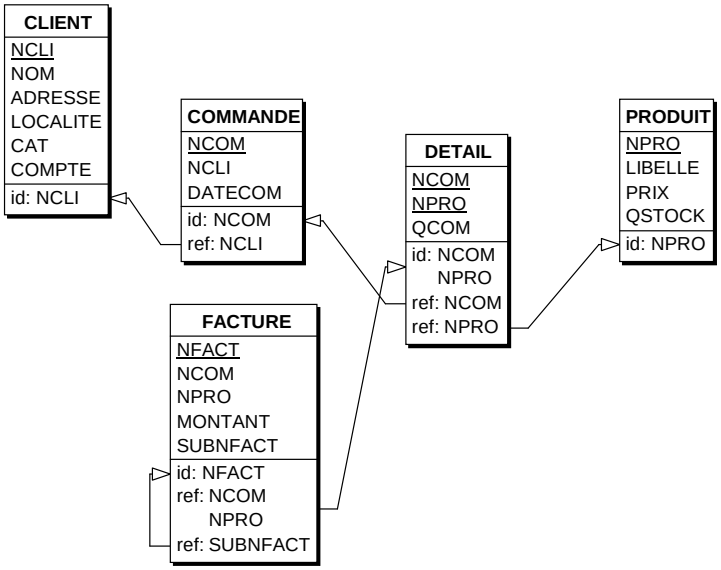


Figure A18.25 - Schema de la base de données CLICOM-proto.db

Cet exercice est extrêmement intéressant, mais va nous paraître également frustrant. D'une part, le code est relativement long (près de 800 lignes, y compris les commentaires) et répétitif. D'autre part, sa modification suite aux rectifications successives des spécifications sera vraisemblablement laborieuse, ce qui réduira l'intérêt de la technique du prototypage. Ce développement est couvert par les sections A18.3.1 à A18.3.10 ci-après.

Dans un deuxième temps, grâce à l'expertise acquise pendant le développement manuel, nous envisagerons de produire cet gestionnaire prototype de manière automatique à partir du schéma des tables. Nous construirons donc un générateur de prototype.

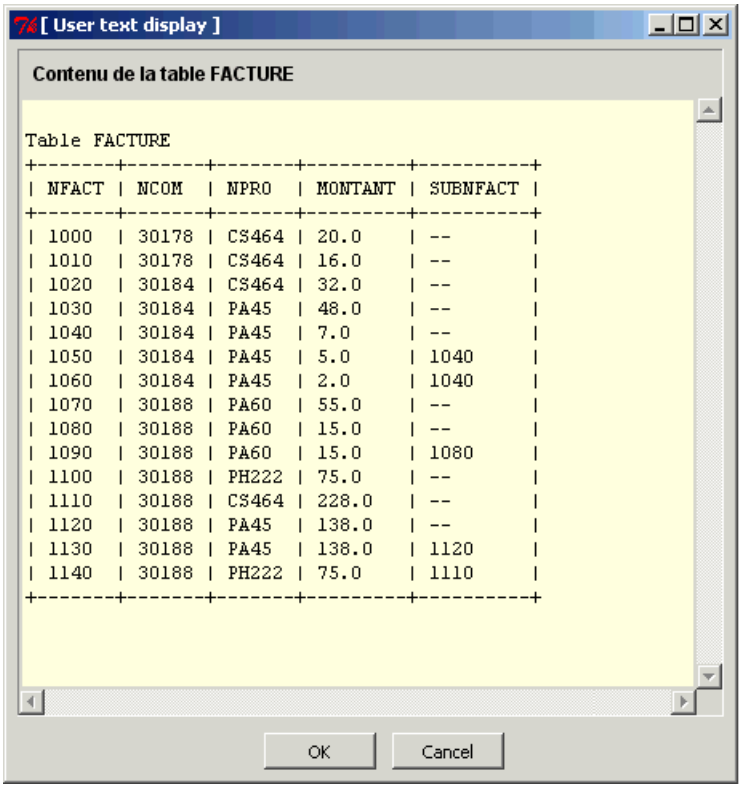


Figure A18.26 - Contenu de la nouvelle table FACTURE

Nous réaliserons ainsi, grâce à cet outil, un environnement simple et efficace de validation d'un schéma conceptuel quelconque. Les remarques des utilisateurs se traduisent exclusivement par des modifications du schéma conceptuel, la génération du nouveau prototype (schéma SQL et interface) n'exigeant que quelques secondes. La construction du générateur est brièvement décrite dans la section A18.3.11.

La section de conclusion A18.3.14 propose quelques pistes d'extension et d'amélioration du modèle de prototype et de son générateur.

A18.3.1 Le panneau de commande du gestionnaire

Le gestionnaire interactif offre à l'utilisateur cinq fonctions lui permettant de créer et manipuler les données de la base de données. Ces fonctions sont accessibles via le panneau de commande de la figure A18.27. Pour chaque table, il est possible de **créer** des lignes, **modifier** des lignes, **supprimer** des lignes, **lister** le contenu de la table et **explorer** les lignes d'une table et les lignes qui leur sont associées.

Nous étudierons chacune de ces opérations en détail.

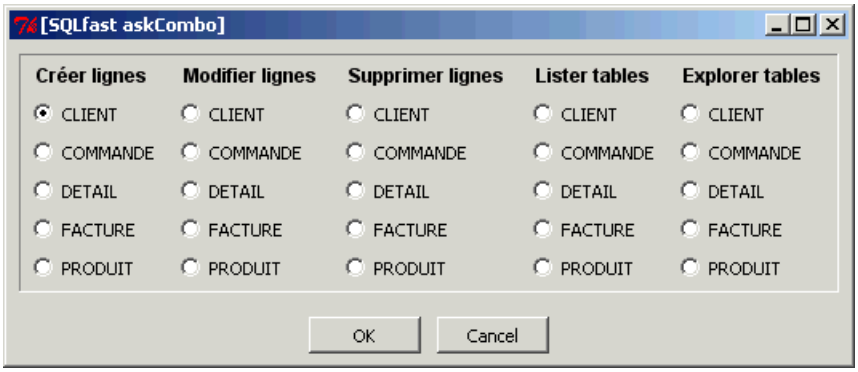


Figure A18.27 - Le panneau de commande du gestionnaire - Première version

A18.3.2 Création dans une table simple sans clé étrangère : CLIENT

La collecte des données d'une ligne de CLIENT se fera via la boîte de saisie de la figure A18.28. Les champs se présentent en colonne et l'étiquette du champ correspondant à une colonne obligatoire est munie d'un caractère '*****'. Le bouton OK confirme la saisie et le bouton Cancel annule la saisie. Un message court (**Créer CLIENT**) indique le fonction de la boîte.

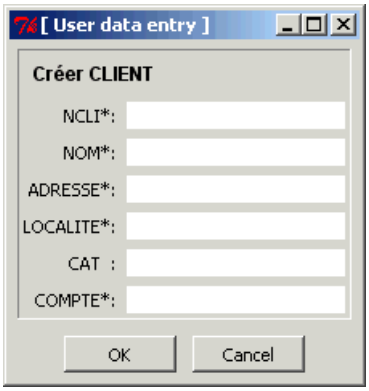


Figure A18.28 - boîte de saisie des données d'une ligne de COMMANDE

Le script de la figure A18.29 réalise l'insertion d'une ligne composée des données saisie par cete boîte.

Une remarque importante dans la perspective de la génération automatique des gestionnaires : les libellés des champs apparaissant dans la boîte de saisie ne sont rien d'autre que les noms des champs. De même, les variables de réception des données saisies sont nommées des noms des colonnes qu'elles représentent `NCLI`, `NOM`, ..., `CAT`, `COMPTE`.

La boîte est créée par l'instruction **ask**. Celle-ci spécifie les variables de réception, le message d'introduction (**[/bCréer CLIENT]**), qui doit apparaître en caractères gras (**/b**), et la définition des champs, chacun précédé de son étiquette. Les séparateurs d'étiquettes (**|**) indiquent que les champs sont disposés verticalement en une seule colonne.

Les valeurs à insérer par la requête **insert** sont celles des variables de réception. Toutes sont munies d'apostrophes, sauf la dernière, qui est un nombre.

```
ask NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE = [/bCréer CLIENT]
    NCLI*:|NOM*:|ADRESSE*:|LOCALITE*:|CAT  :|COMPTE*:;
insert into CLIENT values(
   '$NCLI$','$NOM$','$ADRESSE$','$LOCALITE$','$CAT$',$COMPTE$);
```

Figure A18.29 - Insertion d'une ligne à partir des données fournies par l'utilisateur

Le script A18.30 est plus complet. Il identifie le bouton utilisé pour quitter la boîte (variable système **DIALOGbutton**), et stoppe l'action si ce bouton est Cancel.

```
ask NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE = [/bCréer CLIENT]
    NCLI*:|NOM*:|ADRESSE*:|LOCALITE*:|CAT  :|COMPTE*:;
if ('$DIALOGbutton$' = 'Cancel') stop;
insert into CLIENT values(
   '$NCLI$','$NOM$','$ADRESSE$','$LOCALITE$','$CAT$',$COMPTE$);
```

Figure A18.30 - Prise en compte du bouton **Cancel**

Si une erreur se produit lors de l'insertion de la ligne, un code est renvoyé dans la variable système **SQLdiag**. Ce code est OK si l'opération s'est effectuée avec succès. Dans ce cas, on confirme l'opération (**commitDB**). En cas d'erreur, la procédure `traiterErreurSQL.sql` en informera l'utilisateur.

```
ask NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE = [/bCréer CLIENT]
    NCLI*:|NOM*:|ADRESSE*:|LOCALITE*:|CAT  :|COMPTE*:;
if ('$DIALOGbutton$' = 'Cancel') stop;
insert into CLIENT values(
   '$NCLI$','$NOM$','$ADRESSE$','$LOCALITE$','$CAT$',$COMPTE$);
if ('$SQLdiag$' <> 'OK');
   execSQL traiterErreurSQL.sql;
else;
   commitDB;
endif;
```

Figure A18.31 - Gestion des erreurs SQL

Rappelons que les valeurs fournies par la boîte de dialogue sont des chaînes de caractères[17]. Occupons-nous à présent de la validité des données qui vont être introduire dans la table CLIENT. Il est inutile de vérifier a priori que chaque donnée est du

type de sa colonne puisque le moteur SQL va s'en charger, renvoyant le code d'erreur approprié en cas de violation. Quant aux valeurs des colonnes *not null*, elles réclament un peu de réflexion. Lorsqu'un champ de saisie est laissé vide par l'utilisateur (il n'y a rien introduit, même pas un espace), la variable de réception contient une chaîne vide. On décide de traduire cette absence de valeur par l'indicateur *null*. Nous allons donc remplacer toute valeur *vide* par la chaîne `null`, au moyen d'instructions du type :

```
if ('$ADRESSE$' = '');
  set ADRESSE = null;
else;
  set ADRESSE = '$ADRESSE$';
endif;

if ('$COMPTE$' = '') set COMPTE = null;
```

Si la valeur de la variable est une chaîne vide, celle-ci est remplacée par `null`. Sinon, elle reste inchangée pour les colonnes numériques ou entourée d'apostrophes, pour les colonnes non numériques. La requête insert s'écrit dès lors comme suit :

```
insert into CLIENT values($NCLI$,$NOM$,$ADRESSE$,
                          $LOCALITE$,$CAT$,$COMPTE$)
```

Le soin de vérifier si la colonne laissée sans valeur peut admettre la valeur *null* est confié au moteur SQL, qui informera l'utilisateur en cas de violation.

Cette approche conduit cependant à des scripts relativement complexes dès que le nombre de colonnes est important. D'autre part, le moteur SQL est capable d'exécuter cette transformation bien plus efficacement que l'interpréteur SQLfast.

Procédons dès lors comme suit. La valeur introduite dans la requête `insert` est une expression `case-end` qui renvoie la valeur de la variable de réception entourée d'apostrophes si cette valeur est non vide, et `null` si cette valeur est vide :

```
case when '$ADRESSE$' = '' then null else '$ADRESSE$' end
```

Pour une colonne numérique, on pourrait proposer cette formule :

```
case when '$COMPTE$' = '' then null else $COMPTE$ end
```

Malheureusement, cette formulation ne convient pas dans le cas des valeurs numériques. En effet, lorsque la variable COMPTE n'a pas reçu de valeur, la clause **`else`** ne mentionne plus rien en raison de l'absence d'apostrophes et devient donc incorrecte, ce qui provoque une erreur syntaxique SQL. On traitera donc les colonnes numéri-

17. D'ailleurs, la valeur de toute variable SQLfast est une chaîne de caractères Unicode (à l"exception des manipulations multimédia). Son type dépend de l'opération dans laquelle elle apparaît.

ques séparément, comme illustré dans le script A18.32 pour la colonne COMPTE. Ce script est en outre structuré en une boucle qui permet, d'une part, de corriger les données refusées[18], et d'autre part d'insérer une série de lignes.

```
while (True);
   ask NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE = [/bCréer CLIENT]
       NCLI*:|NOM*:|ADRESSE*:|LOCALITE*:|CAT  :|COMPTE*:;
   if ('$DIALOGbutton$' = 'Cancel') exit;
   if ('$COMPTE$' = '') set COMPTE = null;
   insert into CLIENT values(
   case when '$NCLI$' = ''     then null else '$NCLI$'     end,
   case when '$NOM$' = ''      then null else '$NOM$'      end,
   case when '$ADRESSE$' = ''  then null else '$ADRESSE$'  end,
   case when '$LOCALITE$' = '' then null else '$LOCALITE$' end,
   case when '$CAT$' = ''      then null else '$CAT$'      end,
   $COMPTE$);
   if ('$SQLdiag$' <> 'OK');
      execSQL traiterErreurSQL.sql;
   else;
      commitDB;
   endif;
endwhile;
```

Figure A18.32 - L'assignation des valeurs *null* est confiée au moteur SQL. La procédure est rendue itérative.

Il nous reste encore un petit problème à régler relatif encore une fois aux discordances entre boîte de dialogue et SQL. Lorsqu'une chaîne de caractères est saisie puis rangée dans une variable de réception, le contenu de celle-ci est ensuite injecté tel quel dans l'instruction `insert`. Si cette valeur contient des apostrophes, comme l'adresse du client `B512` (`rue de l'Eté`), elle ne sera conforme à la syntaxe SQL que lorsqu'on les aura doublés ces apostrophes : `rue de l''Eté`. Pour effectuer cette normalisation, nous ferons appel à la fonction SQLquote2 de la librairie LStr :

```
function ADRESSE = LStr:SQLquote2 {$ADRESSE$}
```

Cette function range dans la variable cible (`ADRESSE`) la valeur de son argument (`$ADRESSE$`) dans laquelle les apostrophes ont été doublées. L'argument est placé entre accolades de manière à protéger les virgules éventuelles, qui pourraient être confondues avec les séparateurs d'arguments.[19] On obtient ainsi la version finale de la procédure d'insertion (script A18.33).

18. Pour représenter les données refusées, il conviendrait d'utiliser la forme `ask-u` (`u` pour *update*) au lieu de `ask`. Nous y reviendrons dans les opérations de modification.
19. On pourrait aussi utiliser la fonction SQLfast native `doubleQuote` dans l'instruction `compute ADRESSE = doubleQuote("$ADRESSE$")`. On doit ici utiliser la syntaxe SQL, ce qui nous conduit à entourer la valeurs de guillemets (et pas d'apostrophes bien sûr !)

```
while (True);
   ask NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE = [/bCréer CLIENT]
       NCLI*:|NOM*:|ADRESSE*:|LOCALITE*:|CAT  :|COMPTE*:;
   if ('$DIALOGbutton$' = 'Cancel') exit;
   function NCLI =     LStr:SQLquote2 {$NCLI$};
   function NOM =      LStr:SQLquote2 {$NOM$};
   function ADRESSE =  LStr:SQLquote2 {$ADRESSE$};
   function LOCALITE = LStr:SQLquote2 {$LOCALITE$};
   function CAT =      LStr:SQLquote2 {$CAT$};
   if ('$COMPTE$' = '') set COMPTE = null;
   insert into CLIENT values(
   case when '$NCLI$' = ''     then null else '$NCLI$'     end,
   case when '$NOM$' = ''      then null else '$NOM$'      end,
   case when '$ADRESSE$' = ''  then null else '$ADRESSE$'  end,
   case when '$LOCALITE$' = '' then null else '$LOCALITE$' end,
   case when '$CAT$' = ''      then null else '$CAT$'      end,
   $COMPTE$);
   if ('$SQLdiag$' <> 'OK');
      execSQL traiterErreurSQL.sql;
   else;
      commitDB;
   endif;
endwhile;
```

Figure A18.33 - Les valeurs à insérer sont normalisées selon la syntaxe des constantes SQL

A18.3.3 Insertion dans une table comportant une clé étrangère simple : COMMANDE

La table COMMANDE comporte une clé étrangère mono-composant, NCLI, vers la table CLIENT. On peut traiter ce cas comme celui de la table CLIENT, en invitant l'utilisateur à introduire à la main la valeur de NCLI. En cas d'erreur, le moteur SQL signalera une erreur de violation de l'intégrité référentielle. Une solution beaucoup plus conviviale serait de laisser l'utilisateur sélectionner une des valeurs de NCLI présentes dans la table CLIENT. L'introduction de la valeur serait ainsi rendue plus facile et ne provoquerait pas d'erreur. Pour ce faire, nous associerons au champ NCLI une *liste de valeurs prédéfinies* extraites de la table CLIENT par la requête :

```
select NCLI from CLIENT order by NCLI
```

L'instruction de création de la boîte de saisie est donc la suivante :

```
ask NCOM,NCLI,DATECOM = [/bCréer COMMANDE]
    NCOM*:
   |NCLI*:[!select NCLI from CLIENT order by NCLI]
   |DATECOM*:;
```

Le symbole **!** en préfixe de la requête `select` interdit à l'utilisateur d'introduire une valeur manuellement et l'oblige à sélectionner une valeur dans la liste. On présente ainsi à l'utilisateur la boîte de saisie de la figure A18.34.

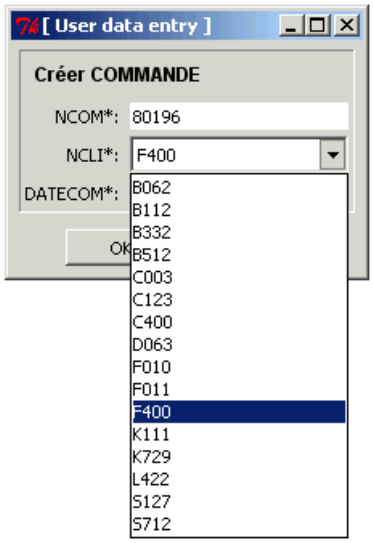


Figure A18.34 - boîte saisie d'une ligne de COMMANDE avec valeurs prédéfinies

Le script A18.35 est celui de l'insertion d'une ligne de DETAIL. On y observe la définition de deux champs à valeurs prédéfinies, correspondant aux deux clés étrangères de la table DETAIL : NCOM et NPRO.

```
while (True);
   ask NCOM,NPRO,QCOM = [/bCréer DETAIL]
       NCOM*:[!select NCOM from COMMANDE order by NCOM]
      |NPRO*:[!select NPRO from PRODUIT order by NPRO]
      |QCOM*:;
   if ('$DIALOGbutton$' = 'Cancel') exit;

   if ('$QCOM$' = '') set QCOM = null;
   function NCOM = LStr:SQLquote2 {$NCOM$};
   function NPRO = LStr:SQLquote2 {$NPRO$};

   insert into DETAIL values(
          case when '$NCOM$' = '' then null else '$NCOM$' end,
          case when '$NPRO$' = '' then null else '$NPRO$' end,
          $QCOM$);
   if ('$SQLdiag$' <> 'OK');
      etc.
   endif;
endwhile;
```

Figure A18.35 - Procédure d'insertion d'une ligne de DETAIL.

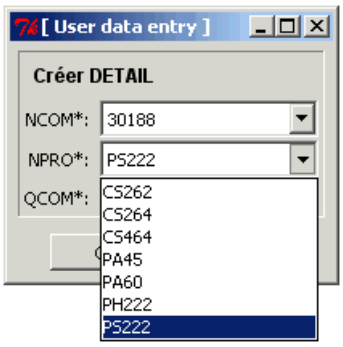


Figure A18.36 - boîte de saisie d'une ligne de DETAIL comportant deux listes de valeurs prédéfinies.

A18.3.4 Insertion dans une table comportant une clé étrangère composite : FACTURE

Pour étudier le cas des clés étrangères composites, nous utilisons la nouvelle table FACTURE référençant la table DETAIL. On y enregistre les données sur les factures. Chacune d'elles référence un détail, plusieurs factures pouvant référencer le même détail. Cette table est donc munie d'une clé étrangère composée des colonnes NCOM et NPRO (figure A18.25). En outre, on lui ajoute une clé étrangère cyclique, SUBNFACT, qui indique, pour chaque facture, la facture qu'elle remplace. Une facture peut donc être remplacée par plusieurs autres factures. SUBNFACT est évidemment une colonne facultative.

La saisie de la valeur de la clé étrangère (NCOM, NPRO) est ici un peu plus délicate que celle de NCLI de COMMANDE. La solution la plus simple serait de définir deux champs de saisie distincts, l'un pour NCOM et l'autre pour NPRO, chacun à liste de valeurs prédéfinies.

Elle souffre cependant d'un inconvénient pour l'utilisateur, qui doit connaître la combinaison valide de (NCOM, NPRO) qui identifie précisément le détail concerné. Il serait bien plus utile, pour éviter tout risque d'erreur, de proposer à l'utilisateur la liste des couples (NCOM, NPRO) effectivement présents dans la table DETAIL. L'instruction ci-dessous crée une boîte de saisie à quatre champs dont le deuxième collecte la combinaison (NCOM,NPRO) sélectionnée dans la liste de valeurs. Cette valeur est assignée à la variable NCOM-NPRO, dont le nom est obtenu par concaténation des noms de ses composants. Les valeurs de ce couple sont obtenus par concaténation des valeurs élémentaires séparées par le caractère `'-'`.[20] La saisie de la valeur de SUBNFACT se fait de la manière habituelle.

```
ask NFACT,NCOM-NPRO,MONTANT,SUBNFACT = [/bCréer FACTURE]
    NFACT*:
   |NCOM-NPRO*:[select NCOM||'-'||NPRO from DETAIL]
```

20. En faisant l'hypothèse, à vérifier, que la nature des valeurs de NCOM et de NPRO ne conduit pas à des ambiguïtés.

```
|MONTANT   :
|SUBNFACT  :[select NFACT from FACTURE order by NFACT]
```

Le résultat est la boîte de la figure A18.37

.

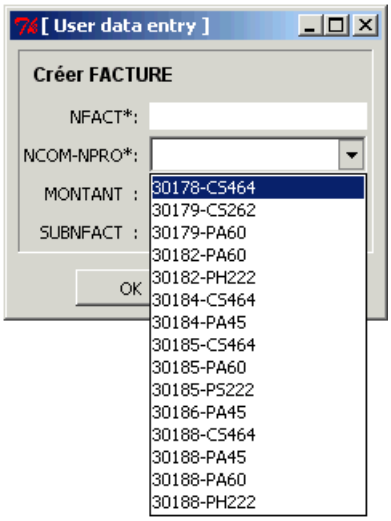


Figure A18.37 - Les champs NCOM et NPRO de la clé étrangère sont combinés pour former un champ unique

Nous ajouterons deux améliorations :

– la concaténation de valeurs de types différents n'est pas acceptée par tous les SGBD. Par prudence, on remplacera son expression par ceci :

```
cast(NCOM as char)||'-'||cast(NPRO as char)
```

– l'utilisateur appréciera que les valeurs soient regroupées par mêmes valeurs de NCOM. On ajoute donc :

```
order by NCOM,NPRO
```

Pour insérer les valeurs de NCOM et NPRO dans la ligne de facture, nous devons redécomposer l'assemblage saisi. Le plus simple est de les extraire de la ligne référencée :

```
extract NCOM,NPRO
      = select NCOM,NPRO from DETAIL
        where cast(NCOM as char)||'-'||cast(NPRO as char)
              = '$NCOM-NPRO$';
```

Le script complet est illustré à la figure A18.38.

```
while (True);
   ask NFACT,NCOM-NPRO,MONTANT,SUBNFACT = [/bCréer FACTURE]
       NFACT*:
      |NCOM-NPRO*:[select cast(NCOM as char)||'-'||
                          cast(NPRO as char)
                  from DETAIL order by NCOM,NPRO]
      |MONTANT  :
      |SUBNFACT  :[select NFACT from FACTURE order by NFACT];
   if ('$DIALOGbutton$' = 'Cancel') exit;
   extract NCOM,NPRO = select NCOM,NPRO from DETAIL
                       where cast(NCOM as char)||'-'||
                             cast(NPRO as char)
                             = '$NCOM-NPRO$';
   function NCOM = LStr:SQLquote2 {$NCOM$};
   function NPRO = LStr:SQLquote2 {$NPRO$};
   if ('$NFACT$' = '')    set NFACT =    null;
   if ('$MONTANT$' = '')  set MONTANT =  null;
   if ('$SUBNFACT$' = '') set SUBNFACT = null;
   insert into FACTURE values($NFACT$,
          case when '$NCOM$' = '' then null else '$NCOM$' end,
          case when '$NPRO$' = '' then null else '$NPRO$' end,
          $MONTANT$,$SUBNFACT$);
   if ('$SQLdiag$' <> 'OK');
      etc.
   endif;
endwhile;
```

Figure A18.38 - Procédure d'insertion d'une ligne de FACTURE.

A18.3.5 Modification de lignes d'une table

Nous traiterons le cas de la table DETAIL. La modification se fait en quatre étapes :

- sélection de la ligne à modifier
- acquisition des valeurs actuelles de cette ligne
- présentation de ces valeurs à l'utilisateur, invité à les modifier
- modification de la ligne.

Le script est détaillé à la figure A18.39.

```
ask id = [/bModifier DETAIL]
          DETAIL:[select cast(NCOM as char)||'-'
                        ||cast(NPRO as char)
                  from DETAIL order by NCOM,NPRO];
if ('$DIALOGbutton$' = 'Cancel') exit;

function id = LStr:SQLquote2 {$id$};

extract NCOM,NPRO,QCOM
      = select NCOM,NPRO,QCOM from DETAIL
        where cast(NCOM as char)||'-'||cast(NPRO as char)='$id$';

ask-u NCOM,NPRO,QCOM = [/bModifier DETAIL]
                NCOM*:[select NCOM from COMMANDE order by NCOM]
               |NPRO*:[select NPRO from PRODUIT order by NPRO]
               |QCOM*:;
if ('$DIALOGbutton$' = 'Cancel') exit;

function NCOM = LStr:SQLquote2 {$NCOM$};
function NPRO = LStr:SQLquote2 {$NPRO$};
if ('QCOM' = '') set QCOM = null;

update DETAIL
set NCOM = case when '$NCOM$' = '' then null else '$NCOM$' end,
    NPRO = case when '$NPRO$' = '' then null else '$NPRO$' end,
    QCOM = $QCOM$
where cast(NCOM as char)||'-'||cast(NPRO as char) = '$id$';

if ('$SQLdiag$' <> 'OK');
   etc.
endif;
```

Figure A18.39 - Modification d'une ligne de la table DETAIL

A18.3.6 Suppression de lignes d'une table

Tout comme pour la modification, l'utilisateur est invité à sélectionner la ligne de la table à supprimer. Par sécurité, on affiche les données de la ligne sélectionnée et on demande à l'utilisateur de confirmer ou d'annuler l'opération (script A18.40).

```
ask id = [/bSupprimer DETAIL]
    DETAIL:[select cast(NCOM as char)||'-'||cast(NPRO as char)
            from DETAIL order by NCOM,NPRO];
if ('$DIALOGbutton$' = 'Cancel') exit;

function id = LStr:SQLquote2 {$id$};

extract NCOM,NPRO,QCOM
  = select NCOM,NPRO,QCOM from DETAIL
    where cast(NCOM as char)||'-'||cast(NPRO as char) = '$id$';

showData NCOM,NPRO,QCOM = [/bConfirmer/annuler la suppression]
         NCOM:|NPRO:|QCOM:;
if ('$DIALOGbutton$' = 'Cancel') exit;
delete from DETAIL
where cast(NCOM as char)||'-'||cast(NPRO as char) = '$id$';

if ('$SQLdiag$' <> 'OK');
   etc.
endif;
```

Figure A18.40 - Suppression de lignes de la table DETAIL

A18.3.7 Liste des lignes d'une table

Cette opération est très simple. Elle a pour effet de présenter dans une fenêtre de texte le contenu de la table sélectionnée (figure A18.42).

Le script est présenté à la figure A18.41. Les données produites sont déviées vers la variable **`resultat`** (suffixée par **`.var`**, pour signifier qu'il s'agit d'une variable). Ensuite un titre est écrit suivi de l'insertion des données sous forme tabulaire. Le contenu de la variable **`resultat`** est affiché dans une fenêtre de texte (`showText`), munie d'un champ de message en gras. Le paramètre **`/w0`** indique l'absence de passage à la ligne lorsque le tableau est trop large. Dans ce cas, on utilisera la barre de défilement horizontale.

```
outputOpen resultat.var;

write Table CLIENT;
select * from CLIENT;
showText resultat = [/w0/bContenu de la table CLIENT];

outputOpen console;
```

Figure A18.41 - Liste des lignes de la table PRODUIT

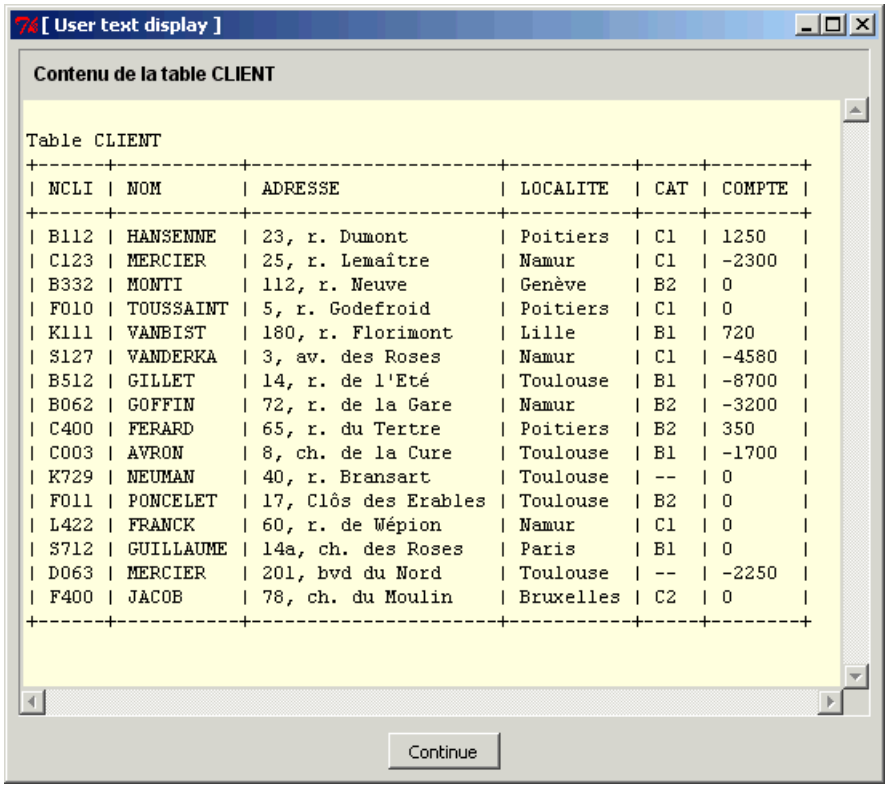


Figure A18.42 - Affichage du contenu de la table CLIENT

Il serait intéressant d'apporter quelques améliorations à ce résultat, par exemple,

- ajouter le tableau de données produit au contenu précédent de la fenêtre au lieu de remplacer ce contenu,
- personnaliser la taille de la fenêtre.

Nous y reviendrons plus loin.

A18.3.8 Exploration du contenu de la base de données

Cette fonction permet de *naviguer* dans les données, structurées en un graphe grâce aux clés étrangères. Les données de la ligne sélectionnée **L** sont présentées dans un format similaire à celui des opérations de création et de modification, mais ces données sont complétées par celles des lignes référencées par **L** et les données des lignes qui référencent **L**. La boîte de la figure A18.43 affiche les données de la commande **30186**, celles du client **C400** qui a passé cette commande et celles des détails (ici un seul) de cette commande.

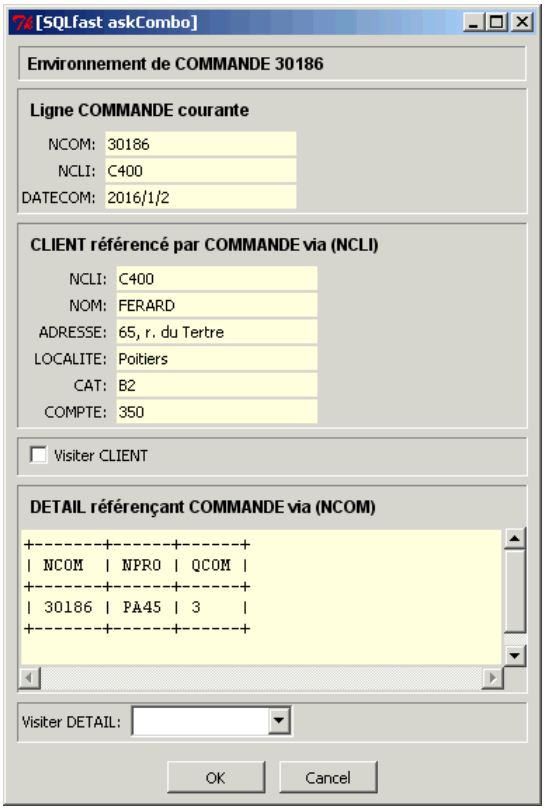


Figure A18.43 - Disposition verticale (en une seule colonne) de le fenêtre d'exploration

Si le nombre de tables voisines est trop important, on adoptera une disposition en trois colonnes : à gauche la ligne sélectionnée, au centre la colonne des lignes référencées et à droite la colonne des lignes référençantes (figure A18.44).

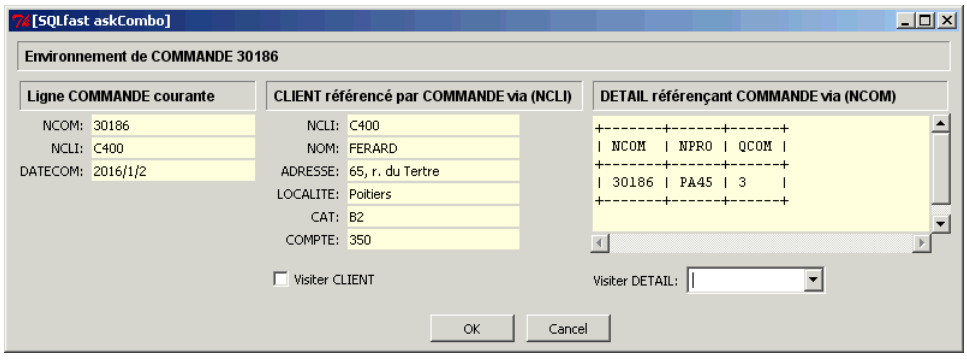


Figure A18.44 - Disposition horizontale (en trois colonne) de le fenêtre d'exploration

La boîte d'affichage des données va également nous servir à *naviguer* de ligne en ligne. Pour chaque table voisine (ici CLIENT et DETAIL), l'utilisateur peut demander l'affichage d'une des lignes affichées, via une case à cocher pour les lignes référencées et par une liste déroulante pour les lignes référençantes. L'exploration se poursuit alors par l'examen de la nouvele ligne ainsi sélectionnée et de son voisinage. Le bouton Cancel permet de stopper la navigation et de revenir au panneau de commande.

La construction des boîtes d'exploration est évidemment plus complexe que celles des autres opérations.

Si la structure de chaque boîte élémentaire dédiée à une table voisine ne pose pas de difficulté particulière, leur assemblage demande des précautions nouvelles. En effet, nous devons stocker simultanément dans des variables de réception les données de la ligne courante et celles des lignes voisines. Pour la ligne *courante* et les lignes voisines *référencées*, il doit exister **une variable par colonne**. Pour chaque table *référençante*, **une variable** (de texte) **par table**. Si une table est référencée ou référençante plus d'une fois, la règle de dénomination des variables par le nom des colonnes associées conduit à des doublons. Pour lever les ambiguïtés potentielles, nous adoptons les règles de dénomination suivantes :

- Pour la **ligne courante**, les variables portent le nom de la colonne correspondante.
- Pour chacune des **lignes référencées**, les variables portent le nom de la colonne correspondante, suffixée par le numéro identifiant de la clé étrangère servant à accéder à cette ligne; ce numéro est fourni par la table du catalogue SYS_KEY.
- Pour les **lignes référençantes**, les données sont rangées dans une variable portant le nom de la table référençante, suffixée par le numéro identifiant de la clé étrangère servant à accéder à la ligne courante.

A18.3.9 Gestion des erreurs

La plupart des erreurs de données liées aux contraintes du schéma sont détectées par le SGBD lors des opérations de création, de modification et de suppression, et capturée par le gestionnaire interactif. Celui-ci appelle la procédure **traiterErreurSQL.sql**, qui analyse les informations du SGBD et affiche un diagnostic. Cette procédure dispose des registres `opération` et `tableCourante`, garnis avant chaque opération. La version de base de la procédure associée au gestionnaire est assez *rustique* mais peut être affinée de manière à informer plus en détail l'utilisateur sur les causes des erreurs.

Il est à noter que les erreurs relatives à l'intégrité référentielle dans les opérations de création et de modification sont évitées grâce à l'usage des listes de valeurs prédéfinies.

A18.3.10Le tableau de commande d'un prototype

La figure A18.45 montre une version plus avancée du tableau de commande de la base de données **CLICOM-proto.db**. Il comporte cinq compartiments :

– **choix d'une opération** :

pour chaque table, **créer** des lignes, **modifier** des lignes, **supprimer** des lignes, **lister** le contenu de la table et **explorer** les lignes d'une table et les lignes qui leur sont associées.

– **paramètres des fonctions Lister tables** :

Il est possible de limiter le nombre de lignes affichées dans la fenêtre de texte, ainsi que la hauteur et la largeur de cette fenêtre.

– **paramètre de la fonction Explorer tables**

On précise la disposition désirée des boîtes élémentaires : verticale (figure A18.43) ou en trois colonnes (figure A18.44).

– **mode d'exécution des mises à jour des données** :

En **mode test**, les modifications sont temporaires et sont annulées en fin de travail. En **mode réel**, les modifications sont définitives.

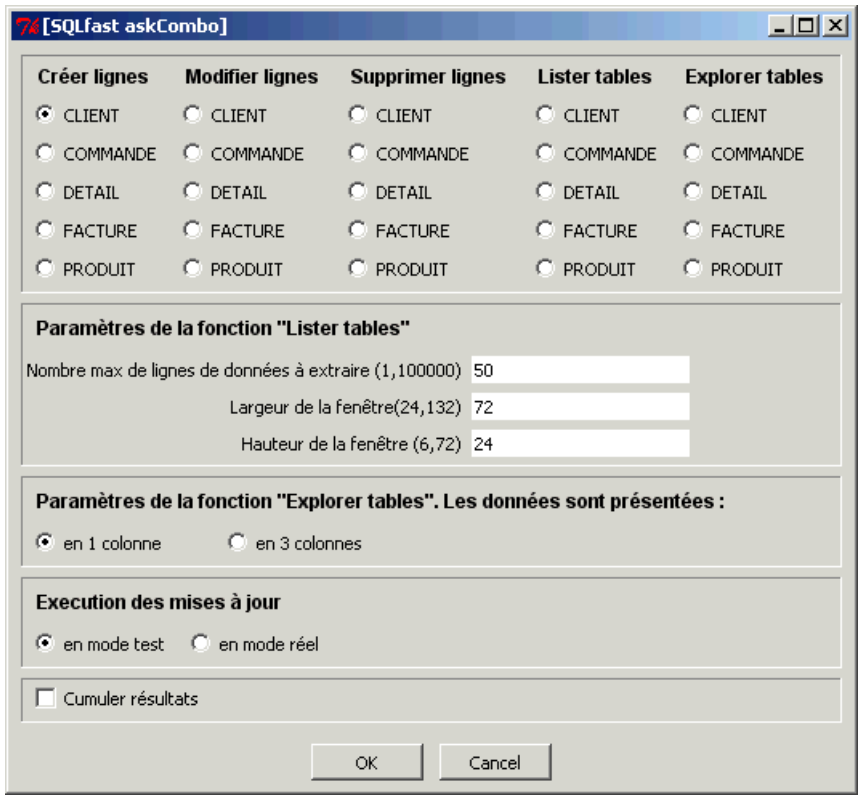


Figure A18.45 - Fenêtre de commande du prototype de la base de données CLICOM-proto.db

– **gestion de la fenêtre de résultat** :

Lorsque les données d'une opération **Lister** sont affichées dans fenêtre de texte, elles peuvent soit remplacer le contenu actuel, soit s'y ajouter.

A18.3.11Générateur de prototype

On observe que la structure d'un gestionnaires suit fidèlement le schéma de sa base de données. D'où l'idée d'en automatiser la construction.

La génération s'appuie sur le contenu du dictionnaire, qui décrit le schéma de la base de données courante. Cependant, nous avons vu que le code d'un prototype comporte de multiples listes de métadonnées et expression SQL plus ou moins complexes dérivant de ce dictionnaire. Reprenons quelques exemples représentatifs rencontrés dans les sections qui précèdent :

```
ask NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE =
    NCLI*:|NOM*:|ADRESSE*:|LOCALITE*:|CAT  :|COMPTE*:;

insert into CLIENT values(
   case when '$NCLI$' = ''      then null else '$NCLI$'     end,
   case when '$NOM$' = ''       then null else '$NOM$'      end,
   case when '$ADRESSE$' = ''   then null else '$ADRESSE$'  end,
   case when '$LOCALITE$' = ''  then null else '$LOCALITE$' end,
   case when '$CAT$' = ''       then null else '$CAT$'      end,
   $COMPTE$);

ask NCOM,NPRO,QCOM = [/bCréer DETAIL]
     NCOM*:[!select NCOM from COMMANDE order by NCOM]
    |NPRO*:[!select NPRO from PRODUIT order by NPRO]
    |QCOM*:;

extract NCOM,NPRO = select NCOM,NPRO from DETAIL
                    where cast(NCOM as char)||'-'||
                             cast(NPRO as char)
                             = '$NCOM-NPRO$';

ask NFACT,NCOM-NPRO,MONTANT,SUBNFACT = [/bCréer FACTURE]
        NFACT*:
       |NCOM-NPRO*:[select NCOM||'-'||NPRO from DETAIL]

update DETAIL
set NCOM = case when '$NCOM$' = '' then null else '$NCOM$' end,
    NPRO = case when '$NPRO$' = '' then null else '$NPRO$' end,
    QCOM = $QCOM$
where cast(NCOM as char)||'-'||cast(NPRO as char) = '$id$';

delete from DETAIL
where cast(NCOM as char)||'-'||cast(NPRO as char) = '$id$';
```

Pour simplifier les algorithmes de génération, ces listes sont précalculées et rangées dans trois tables qui complètent les tables du dictionnaire.

Le générateur est disponible sous la forme du fichier **GenererPrototype.sql**. Son code est largement documenté. Hors commentaires et lignes blanches, il comporte 470 lignes (approximativement 300 instructions SQLfast), réparties comme suit :

- Création des tables additionnelles du dictionnaire : 115 lignes
- Génération du tableau de commande et des aiguillages : 60 lignes
- Génération des sections de création de lignes : 52 lignes
- Génération des sections de modification de lignes : 70 lignes
- Génération des sections de suppression de lignes : 28 lignes
- Génération des sections de listage de tables : 12 lignes
- Génération des sections d'exploration de lignes : 95 lignes
- Diverses opérations de service : 40 lignes.

A18.3.12Application à la base de données ABCDE-proto.db

Le schéma de la base de données ABCDE-proto.db est particulièrement complexe et constitue un véritable *crash test* du générateur de prototype. Il inclut notamment des identifiants primaires composites, des clés étrangères multiples vers la même table et des clés étrangères cycliques multiples. Son schéma conceptuel est présenté à la figure A18.46 et son schéma SQL à la figure A18.47.

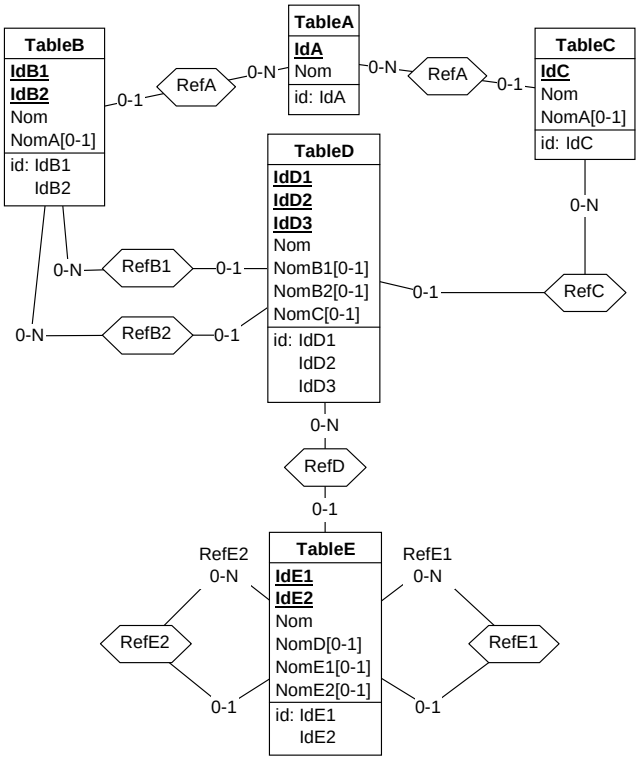


Figure A18.46 - Schema conceptuel de la base de données ABCDE-proto.db

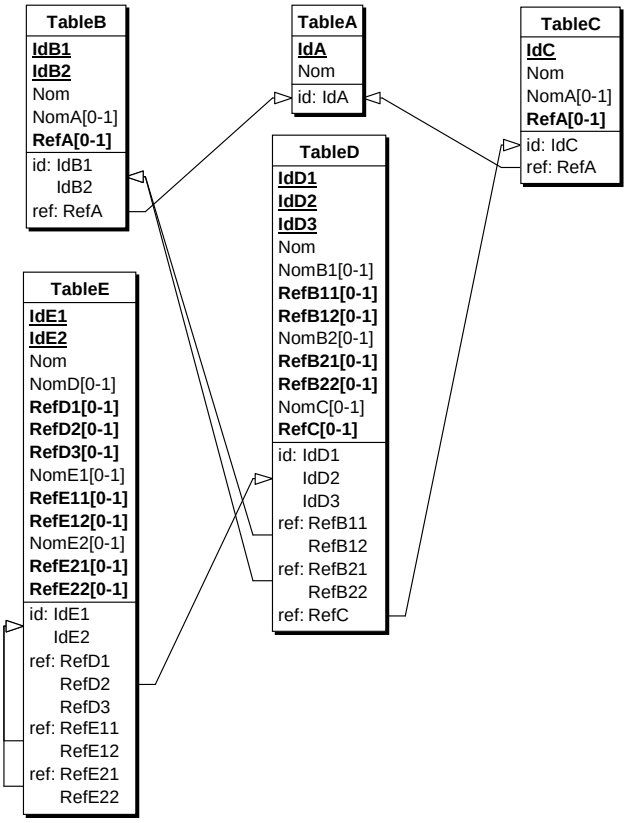


Figure A18.47 - Schema SQL de la base de données ABCDE-proto.db

La figure A18.48 montre un contenu typique des cinq tables de la base de données. Les colonnes composant les identifiants et les clés étrangères sont de nature abstraite. Cependant, pour faciliter la lecture des données de la figure A18.48, chaque table dispose d'une colonne **Nom** dont les valeurs sont uniques, non seulement dans chaque table, mais en plus pour toute la base de données. Ainsi, la valeur **`Pierre`** n'est présente que dans une seule ligne d'une seule table (**TableD**). De même, on a associé à chaque clé étrangère la valeur de Nom de la ligne référencée. Par exemple, la clé étrangère composite (RefD1,RefD2,RefD3) de la table TableE qui référence la table TableD, via son identifiant primaire (IdD1,IdD2,IdD3), est accompagnée dans TableE de la colonne NomD, qui reprend la valeur de Nom de la ligne de TableD référencée.

La base de données SQL et son contenu correspondant à la figure A18.48 est créée par le script SQLfast **Creer-ABCDE.sql**.

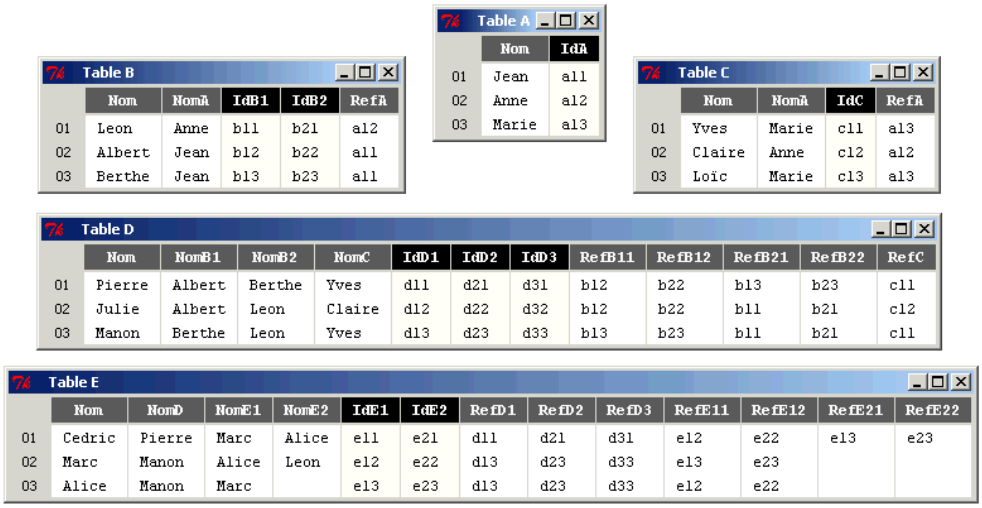


Table A

	Nom	IdA
01	Jean	a11
02	Anne	a12
03	Marie	a13

Table B

	Nom	NomA	IdB1	IdB2	RefA
01	Leon	Anne	b11	b21	a12
02	Albert	Jean	b12	b22	a11
03	Berthe	Jean	b13	b23	a11

Table C

	Nom	NomA	IdC	RefA
01	Yves	Marie	c11	a13
02	Claire	Anne	c12	a12
03	Loïc	Marie	c13	a13

Table D

	Nom	NomB1	NomB2	NomC	IdD1	IdD2	IdD3	RefB11	RefB12	RefB21	RefB22	RefC
01	Pierre	Albert	Berthe	Yves	d11	d21	d31	b12	b22	b13	b23	c11
02	Julie	Albert	Leon	Claire	d12	d22	d32	b12	b22	b11	b21	c12
03	Manon	Berthe	Leon	Yves	d13	d23	d33	b13	b23	b11	b21	c11

Table E

	Nom	NomD	NomE1	NomE2	IdE1	IdE2	RefD1	RefD2	RefD3	RefE11	RefE12	RefE21	RefE22
01	Cedric	Pierre	Marc	Alice	e11	e21	d11	d21	d31	e12	e22	e13	e23
02	Marc	Manon	Alice	Leon	e12	e22	d13	d23	d33	e13	e23		
03	Alice	Manon	Marc		e13	e23	d13	d23	d33	e12	e22		

Figure A18.48 - Exemple de contenu de la base de données ABCDE-proto.db

A18.3.13Les scripts des prototypes

Le répertoire dédié au chapitre 18 contient une collection de scripts illustrant la question des prototypes. En particulier :

- **Creer-CLICOM-proto.sql** : script de création de la base de données CLICOM-proto.db.
- **Creer-ABCDE-proto.sql** : script de création de la base de données ABCDE-proto.db.
- **GenererPrototype.sql** : générateur de prototype
- **TraiterErreurSQL.sql** : procédure de service affichant les informations sur les erreurs détectées ; nécessaire à l'exécution des prototypes.
- **Prototype_CLICOM-proto.sql** : pour les utilisateurs impatients, prototype prégénéré de la base de données CLICOM-proto.db.
- **Prototype_ABCDE-proto.sql** : pour les utilisateurs impatients, prototype prégénéré de la base de données ABCDE-proto.db.

A18.3.14Prototypage d'une base de données : conclusions

L'usage de prototypes est une technique efficace de validation de spécifications conceptuelles. Impliquant de manière active et accessible les utilisateurs, elle doit obtenir une meilleure adhésion de la part de ceux-ci que les autres techniques.

Elle n'est cependant envisageable que si la production de ces prototypes est très largement automatisée. Ce qui exige du formalisme de spécification un haut degré de précision, qui est garanti dans notre cas par la sémantique interne du modèle

Entité-Association. Toute construction d'un schéma peut être traduit complètement et sans ambiguïté par des constructions SQL.

Mentionnons quelques pistes d'extension et d'amélioration du générateur que nous avons construit :

- Ajouter une fonction de *reporting* ou d'*exportation* selon un format d'échange standard.
- Prévoir la définition d'un *filtrage*, sous la forme d'un prédicat SQL ou d'une vue, limitant le champ d'action des opérations à un sous-ensemble des lignes des tables.
- *Migration des données d'un prototype vers le suivant*. Lorsque le schéma est modifié pour satisfaire les remarques des utilisateurs, le schéma SQL doit être modifié en conséquence. Le prototype est alors regénéré automatiquement, mais les données précédemment introduites ne sont généralement plus conformes au nouveau schéma. Si les modifications sont modestes, il est le plus souvent assez facile de rédiger les scripts d'extraction et de chargement des données tenant compte des modifications. Cette rédaction peut elle-aussi être largement automatisée.
- *Présentation des données sous forme d'agrégats*. Les prototypes développés et générés s'articulent exclusivement sur les tables du schéma, la table étant l'unité de gestion et de consultation, comme le montre la fenêtre de commande. Cette unité n'est pas nécessairement la plus naturelle du point de vue des utilisateurs, lesquels voient souvent les données sous la forme d'agrégats plus riches que ceux des tables de base normalisées (on pense par exemple aux documents structurés étudiés au chapitre 18). Les utilisateurs pourraient légitimement demander à voir le nom et l'adresse du client dans l'écran de la commande ou le libellé et le prix du produit dans l'écran d'un détail. Cette extension est aisément réalisable par la technique des vues SQL. Si, pour une table de nom **T**, on crée une vue de nom __**T** qui complète les données de **T**, le prototype d'interface va remplacer les occurrences de **T** par __**T**. Le repérage des colonnes de **T** dans __**T** est un problème assez simple grâce au dictionnaire de SQLfast.
- Insertion initiale de données extraites d'une *base de données existante* ou génération de données synthétiques
- *Prototypage des traitements* via des vues prédéfinies ou de petits scripts SQLfast ; sous forme d'une bibliothèque de fonctions.
- *Prototypage des interfaces utilisateurs*. SQLfast convient particulièrement à la construction rapide de prototypes de dialogues.

A18.4INTÉGRATION DE SCHÉMAS

<Cette section reprend la section 17.9 de la 2e édition, plus étendue que son équivalent 18.9 de la présente édition>

L'analyse du domaine basée sur une décomposition de celui-ci en sous-systèmes homogènes conduit à une collection de sous-schémas corrects, normalisés et validés. Le schéma conceptuel global s'obtient à partir de ceux-ci via le processus d'**intégration**. D'autre part, chaque sous-schéma conceptuel est généralement obtenu par l'intégration de fragments issus de l'analyse de diverses sources indépendantes, textuelles ou formulaires. Dans les organisations d'une certaine taille, il n'est pas rare que chaque sous-système possède sa propre culture et en particulier sa propre manière de percevoir et de nommer les choses. On comprend dès lors que l'intégration de sous-schémas conceptuels qui modélisent des parties identiques du domaine d'application puisse constituer une opération complexe, qui ne se limite pas à *additionner* ces schémas.

L'intégration de schémas, qui est un des problèmes les plus complexes, fait l'objet de nombreux travaux depuis plusieurs dizaines d'années. Il n'existe pas à l'heure actuelle de solution générale, raison pour laquelle nous proposerons ici une description du problème et des pistes de résolution sans prétendre à l'exhaustivité. Son principe est le suivant.

> Étant donné un ensemble **SS** = {$\mathbf{S_1}$, $\mathbf{S_2}$, ..., $\mathbf{S_N}$} de **N** schémas relatifs à **N** sous-systèmes, non nécessairement disjoints, d'un même domaine d'application, l'intégration de **SS** consiste à produire un schéma global unique **S** qui représente, *sans redondance*, *tous les concepts* des schémas de **SS** et *eux seulement*. Tout type de faits représenté dans un schéma $\mathbf{S_i}$ est aussi représenté dans **S**. Tout type de faits représenté dans **S** l'est aussi dans au moins un des sous-schémas de **SS**. Après intégration complète, chaque sous-schéma conceptuel $\mathbf{S_i}$ apparaît comme une vue du schéma global **S**[21].

Nous allons nous heurter à plusieurs difficultés lorsque nous essayerons de détecter les types de faits qui sont représentés simultanément dans deux schémas. En effet, ces types de faits auront souvent des représentations différentes (conflits de nom, conflit syntaxique), mais en outre ils seront parfois perçus de manière différente (conflit sémantique). La procédure d'intégration devra identifier ces conflits et les résoudre.

A18.4.1 Scénarios d'intégration

Sauf cas particuliers (schémas peu nombreux et quasi identiques), on suggère généralement de procéder à l'intégration de SS de manière itérative, en intégrant une paire de schémas à la fois. On réduit ainsi le processus général à une suite d'**intégrations binaires**. La figure A18.49 illustre le scénario que nous retiendrons.

21. En d'autres termes, les données correspondant à un sous-schéma sont calculables à partir des données correspondant au schéma global. Cette propriété apparaîtra de manière concrète dans les sections 18.9 et 20.10.

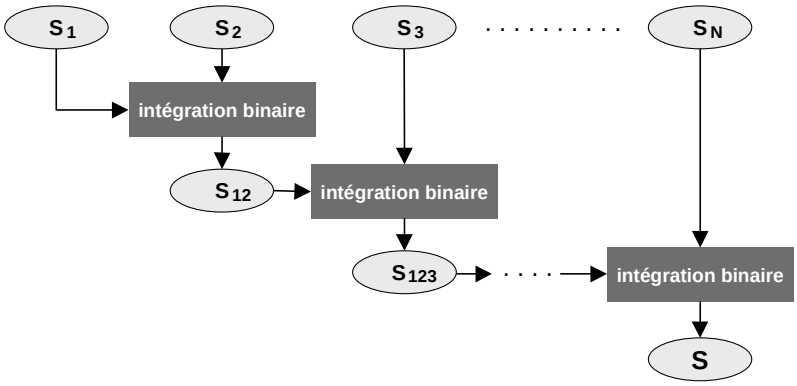


Figure A18.49 - Procédure pratique d'intégration de N schémas en N - 1 étapes

Il consiste à ordonner les schémas par ordre de qualité décroissante, le premier, S_1, étant le schéma le plus riche, le plus expressif, le plus complet et le plus fiable. On intègre ensuite S_1 avec S_2, ce qui produit un premier schéma intégré S_{12}. Le schéma S_3 est alors intégré à S_{12}, ce qui donne S_{123}. En procédant de la sorte, dans cet ordre, jusqu'à épuisement de SS, on obtient le schéma global S.

Symboliquement, en notant $\cup$ le processus d'intégration binaire, on écrira :

$$S_{12} = S_1 \cup S_2;\ S_{123} = S_{12} \cup S_3;\ \ldots.$$
$$S = S_{12\ldots N-1} \cup S_N$$

Une propriété intéressante de cette procédure, dite **en échelle**, est que la séquence $<S_1, S_{12}, S_{123}, S_{12\ldots N-1}, S>$ définit les états successifs, de plus en plus complets, du schéma recherché. Il existe d'autres scénarios d'intégration binaire, notamment le suivant, structuré en arbre binaire équilibré :

$$S_{12} = S_1 \cup S_2;\ S_{34} = S_3 \cup S_4;\ S_{1234} = S_{12} \cup S_{34}$$
$$S_{56} = S_5 \cup S_6;\ S_{78} = S_7 \cup S_8;\ S_{5678} = S_{56} \cup S_{78}$$
$$S_{12345678} = S_{1234} \cup S_{5678};\ \ldots.$$
$$S = S_{1\ldots k-1} \cup S_{k\ldots N}$$

A18.4.2 Principes de l'intégration binaire de schémas

Supposons que les schémas S_1 et S_2 à intégrer soient parfaitement *alignés*, de sorte que tout type de faits représentés dans les deux schémas est modélisé exactement de la même manière : même *nature* d'objet[22] et même *nom*. Dans ce cas, l'intégration se réduit à une simple fusion technique. Ce processus, illustré à la figure A18.50, est particulièrement simple.

22. Type d'entités, type d'associations, domaine de valeurs, attribut ou contrainte.

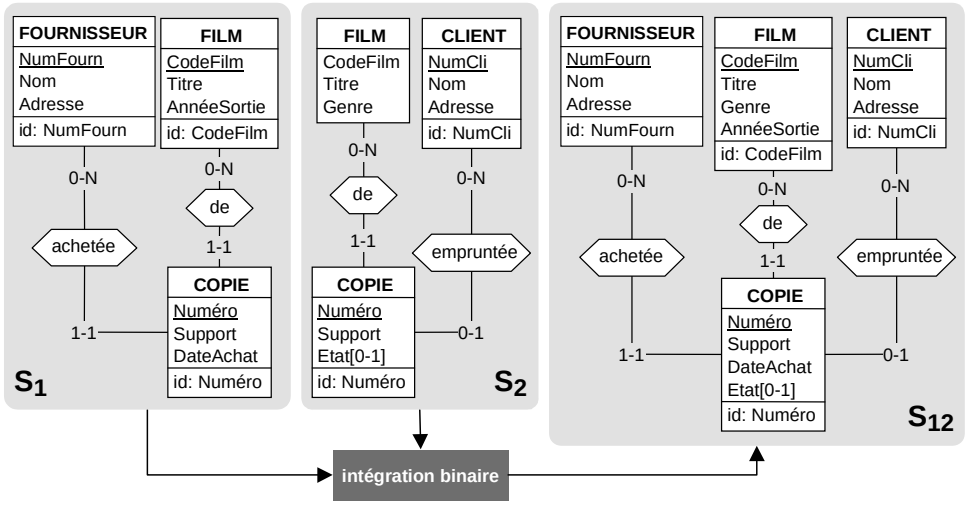


Figure A18.50 - Une intégration réduite à une simple fusion

Malheureusement, les choses ne sont pas toujours aussi simples. Par exemple, un type de faits sera représenté dans un schéma par un attribut et dans l'autre par un type d'entités, sous des noms différents, et soumis à des contraintes différentes, voire contradictoires. Le principe de la fusion que nous venons d'illustrer reste valable dans ce cas, pour autant que nous ayons auparavant *préparé* les deux schémas. La procédure générale d'intégration binaire comporte trois étapes :

1. recherche des correspondances ;
2. unification des schémas ;
3. fusion des schémas.

Nous allons examiner en détail et illustrer chacune de ces étapes, avant de développer une étude de cas complète.

A18.4.3 Recherche des correspondances[23]

On rappelle d'abord que toute instance d'un objet d'un schéma *représente* (ou *dénote*, ou *désigne*) un objet ou un fait du domaine d'application : une entité d'un type représente un objet ou fait réel, une valeur d'attribut représente l'état d'une propriété d'un objet réel et une association représente une relation entre des objets réels. Soit O_1 un objet du schéma S_1; on dénotera par D_1 l'ensemble des objets ou faits du domaine d'application que désignent les instances de O_1.

Une **correspondance** entre deux objets (ou groupes d'objets) O_1 et O_2 de deux schémas S_1 et S_2 à intégrer est un **lien** entre ces objets par lequel on déclare que ceux-ci présentent une **similitude** dont on indique la nature. Plus précisément, cette

23. Processus généralement appelé *Analyse des conflits* dans la littérature [Batini, 1992].

correspondance indique qu'il existe une relation entre les ensembles D_1 et D_2 du domaine. La nature de cette relation peut être très variée, mais nous la limiterons à six types de correspondance positive qui suffiront en pratique : l'*égalité*, la *complémentarité*, la *compréhension*, l'*union*, la *dérivation* et la *dérivation mutuelle*. On y ajoute la correspondance négative de *différence*. Certaines correspondances sont symétriques, les autres étant orientées. Dans ce dernier cas, l'objet à l'origine de la correspondance est qualifié de **dominant**, l'autre étant qualifié de **mineur**. On trouvera aux figures A18.60 et A18.63 des exemples de ces correspondances.

- **Égalité** (=). Une correspondance d'égalité, notée =, existe entre O_1 et O_2 si $D_1 = D_2$. L'égalité indique que les objets O_1 et O_2 ont même sémantique et qu'il est inutile de conserver les deux. Si les objets sont de même nature, l'égalité correspond généralement à une synonymie. Cette correspondance est symétrique.
- **Complémentarité** (cp). Une correspondance de complémentarité, notée cp, existe entre O_1 et O_2 s'il existe une bijection b: $D_1 \longleftrightarrow D_2$, indiquant que chaque objet de D_2 est un *complément* ou *une partie* d'un objet de D_1. Les objets O_1 et O_2 doivent être conservés, unis par un type d'associations 1:1 ou 1:N. Cette correspondance est orientée, l'objet mineur étant considéré comme un complément de l'objet dominant.
- **Compréhension** ($\supseteq$). Une correspondance de compréhension, notée $\supseteq$, existe entre O_1 et O_2 si $D_1 \supseteq D_2$. La compréhension indique généralement que l'objet O_1 doit être considéré comme un surtype de O_2. Cette correspondance est orientée, l'objet dominant représentant un sur-ensemble de celui que représente l'objet mineur.
- **Union** ($\cup$). Une correspondance d'union, notée $\cup$, exprime le fait que, pour les objets en correspondance (au moins deux), il existe un objet O_3 pertinent, désignant l'ensemble D_3, et tel que $D_3 \supseteq D_1$, $D_3 \supseteq D_2$, etc. L'union indique généralement la nécessité de créer un nouvel objet O_3 et de le déclarer surtype de O_1, O_2, etc. Cette correspondance est symétrique.
- **Dérivation** (C). Une correspondance de dérivation, notée C, existe entre O_1 et O_2 s'il existe une fonction de calcul c: $D_1 \rightarrow D_2$. Cette fonction peut être exprimée comme une contrainte d'intégrité ou par une fonction procédurale de forme quelconque. La dérivation suggère de ne pas conserver l'objet O_2 dans le schéma intégré. Cette correspondance est orientée, les instances de l'objet mineur étant calculables à partir de celles de l'objet dominant.
- **Dérivation mutuelle** (Cm). Une correspondance de dérivation mutuelle, notée Cm, exprime le fait que, si O_1 et O_2 étaient réunis dans le même schéma, leur union ferait l'objet d'un ensemble de contraintes d'intégrité exprimant le fait que les instances de certains composants de O_2 sont dérivables de celles de certains composants de O_1 et/ou inversement. La dérivation mutuelle peut se réduire à une dérivation simple (C) si on enrichit un des schémas de certains composants de l'autre. Cette correspondance symétrique est souvent complexe à traiter.
- **Différence** ($\neq$). Une correspondance de différence, notée $\neq$, existe entre O_1 et O_2 si, *contrairement à ce que suggèrent les schémas*, il n'existe aucune relation

sémantique entre ces objets. La différence sera notamment utilisée lorsque deux objets indépendants portent le même nom (homonymie). Cette correspondance, contrairement aux autres, est *négative* (elle affirme l'inexistence d'une propriété apparente). On se contentera bien sûr d'indiquer les différences qui peuvent prêter à confusion. Cette correspondance est symétrique.

On ajoute quatre règles générales à respecter lors de la définition de correspondances.

- *Règle propre aux attributs* : une correspondance peut être établie entre deux attributs s'il existe une correspondance positive entre leurs parents respectifs, éventuellement via leurs surtypes.
- *Règle propre aux types d'associations* : une correspondance peut être établie entre deux types d'associations s'il existe des correspondances positives entre leurs rôles ou les types d'entités qui jouent ces rôles, éventuellement via leurs surtypes.
- Les deux règles précédentes sont spécifiques aux correspondances homogènes. En ce qui concerne les correspondances hétérogènes, elle doivent être vérifiées après la transformation recommandée en A18.4.4 (*Traitement des correspondances hétérogènes positives*).
- On convient qu'il existe une correspondance d'égalité implicite entre deux objets de même nature, de même nom et, pour des attributs, de parents en correspondance d'égalité. Les correspondances implicites ne sont pas notées (figure A18.50).

Correspondances homogènes et hétérogènes

Une correspondance qui associe des objets de même nature (types d'entités, types d'associations, attributs, etc.) est qualifiée d'*homogène*. Elle est *hétérogène* lorsqu'elle associe des objets de natures différentes : un attribut avec un type d'entités ou avec un rôle, un type d'entités avec un type d'associations. On rencontrera cependant des correspondances hétérogènes qui associent deux attributs. Dans ce cas, cette correspondance ne respecte pas la règle propre aux attributs énoncée précédemment.

Repérage des objets en correspondance

Comment identifier les objets susceptibles d'être en correspondance ? Il existe actuellement trois familles de techniques : l'*analyse des noms*, l'*analyse structurelle* et l'utilisation d'*ontologies*.

a) Analyse des noms

Cette technique, de nature linguistique, est basée sur la comparaison des noms des objets des schémas. Idéalement on recherche les noms identiques (FACTURE et FACTURE). Cette égalité de nom sera élargie aux variantes grammaticales (CLIENT et CLIENTS), aux synonymes (ORDRE et COMMANDE), aux variantes sémantiques (ARTICLE et REPORTAGE), à la proximité sémantique (CLIENT et FOURNISSEUR, COMMANDE et FACTURE), voire à une similitude plus complexe (MONTANT-

TOTAL et PRIX-UNITAIRE. On repérera par la même occasion les fausses égalités, ou homonymes : ARTICLE (en stock) et ARTICLE (d'un journal).

L'analyse se porte d'abord sur les **types d'entités**, dont on recherchera les couples portant le même nom, compte tenu des variantes décrites ci-dessus.

Suit alors l'analyse des **attributs** des types d'entités en correspondance (positive). On admet implicitement que les attributs de parents en correspondance d'égalité, de compréhension ou d'union sont égaux s'ils portent le même nom, et sont différents sinon :

si CLIENT ∪ FOURNISSEUR, *alors* CLIENT.Nom = FOURNISSEUR.Nom.

L'analyse des **types d'associations** procède du même principe : on repère les types d'associations existant entre des types d'entités en correspondance et on analyse leurs noms, en admettant les variations décrites ci-dessus. L'analyse des noms est partiellement automatisable.

b) Analyse structurelle

Cette technique se base sur une similitude des structures. On repère deux objets à examiner dès que leurs liens avec les objets voisins dans leurs schémas respectifs sont, sinon identiques, du moins similaires. Par exemple, si deux types d'entités possèdent des attributs semblables, jouent des rôles similaires et sont soumis à des contraintes analogues, on examinera s'ils peuvent être mis en correspondance. Dans le cas de grands schémas, on pourra utiliser des mesures de similitude. Lors de la comparaison de deux objets, on attribue un score à chaque élément structurel similaire. Si le score total de la paire est supérieur à un seuil, on déclare la paire *en correspondance*. Reste alors à valider cette hypothèse et à qualifier cette correspondance.

Exemple. Considérons la paire de types d'entités {A, B}. Pour toute paire d'**attributs similaires** (mêmes domaines, mêmes cardinalités), on compte un score de 1 ; si les domaines sont différents mais comparables, on se contente de 0,8. Pour toute paire de **rôles similaires** (mêmes cardinalités, dans des types d'associations similaires, rôles opposés joués par des types d'entités similaires ou en correspondance) on compte un score de 2. Si les identifiants primaires sont similaires (composants similaires), on compte 3. Si le score total de la paire {A, B} dépasse 15, alors on déclare ces types d'entités similaires[24].

Cette technique vient en complément de l'analyse des noms. Elle est largement automatisable.

c) Utilisation d'ontologies

Une ontologie est une représentation formalisée de l'ensemble des concepts d'un domaine d'application et de leurs associations. Bien que cette définition soit analogue à celle du schéma conceptuel, l'ontologie relève d'une discipline distincte.

24. Cet exemple est traité de manière simpliste. Par exemple, les mesures de similitude sont souvent normalisées par une réduction entre 0 et 1. Ainsi, le seuil de similitude est une constante qui ne dépend pas du schéma.

L'ontologie, ainsi que toutes les technologies qui lui sont associées, forme la base de ce qu'on appelle le *Web sémantique*. L'intérêt des ontologies dans l'intégration de schémas vient de certaines de leurs propriétés : elle peuvent inclure des méta-types, des types et des instances, elle peuvent décrire des concepts et des associations de manière moins précise (donc plus large et plus générique), des ontologies de référence existent dans de nombreux domaines, il existe des langages dotés d'une sémantique interne (mathématique) précise pour définir des ontologies, il existe des outils de manipulation et d'inférence d'ontologies.
Une ontologie associée à un domaine général (*gestion financière* par exemple) dont notre domaine d'application est un sous-ensemble constitue un modèle de référence utile dans la détection semi-automatique des correspondances. Une ontologie pourra par exemple nous apprendre qu'un EMPLOYE est une PERSONNE et que AUTEUR est un rôle qu'une PERSONNE peut jouer. Elle nous indiquera aussi qu'une PERSONNE a généralement un nom, un prénom et une date de naissance.

Ces connaissances générales peuvent nous guider très utilement dans la recherche des objets en correspondance, notamment en complément de l'analyse des noms et des structures. On notera enfin que la détection des correspondances entre ontologies et l'intégration d'ontologies sont des problèmes très actuels[25], fort proches du processus qui nous occupe dans cette section.

A18.4.4 Unification des schémas[26]

L'unification de deux schémas entre lesquels a été défini un ensemble de correspondances consiste à modifier, si nécessaire, les objets en correspondance de manière à rendre aussi simple que possible le processus ultérieur de fusion. Dans chaque schéma, des objets sont ajoutés, d'autres sont supprimés ou sont transformés. Nous examinerons, pour chaque type de correspondances, les opérations recommandées pour unifier deux schémas en correspondance. Nous devons distinguer les correspondances homogènes et hétérogènes.

A. Traitement des correspondances homogènes

a) Correspondance d'égalité (figure A18.51)

Les deux objets reçoivent le même nom dans les deux schémas. Ce nom est celui d'un des deux objets ou un nouveau nom plus approprié. Si les objets sources sont soumis à des contraintes différentes, on choisira ou calculera la plus générale.

25. Consulter par exemple, comme point d'entrée, le site http://www.ontologymatching.org
26. Processus aussi appelé *Résolution des conflits* dans la littérature.

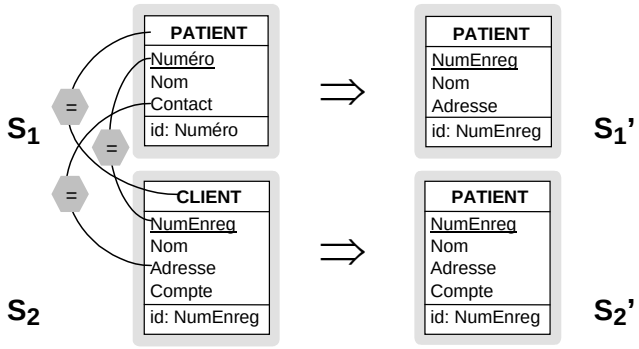


Figure A18.51 - Unification selon une correspondance **=** homogène

Par exemple, si les attributs en correspondance ont des cardinalités respectives [0-5] et [0-8], on choisira [0-8]. Il en ira de même des domaines de valeurs des attributs et de la structure des attributs composés ou encore des cardinalités des rôles des types d'associations. Si les structures sont contradictoires, elle constituent un *conflit sémantique*, qui sera étudié à la section A18.4.6.

b) Correspondance de complémentarité (figure A18.52)

L'objet mineur (complément) est attaché à l'objet dominant. Une copie minimale de celui-ci est introduite dans le schéma du complément et un type d'associations 1:1 ou 1:N est établi entre les types d'entités. On n'effectue pas à ce stade d'élimination de composants redondants dans l'objet mineur (attribut NumEnreg dans HISTORIQUE par exemple). S_1 reste inchangé.

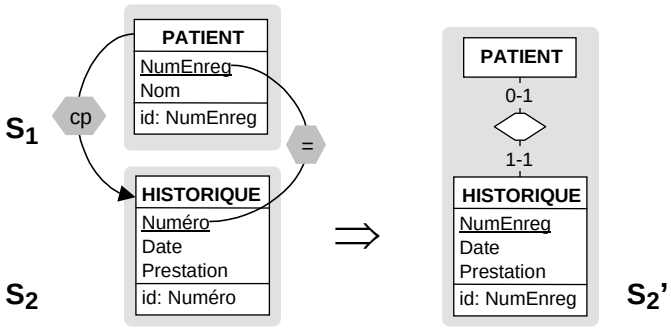


Figure A18.52 - Unification selon une correspondance **cp** homogène (S_1 inchangé)

c) Correspondance de compréhension (figure A18.53)

Une copie minimale du type d'entités dominant est introduite comme surtype dans le schéma du type d'entités mineur. On n'effectue pas à ce stade de migration de composants du sous-type vers le surtype (attributs Matricule et Nom par

exemple). Les types d'associations sont traités de la même manière mais soumis à une contrainte d'inclusion. S_1 reste inchangé.

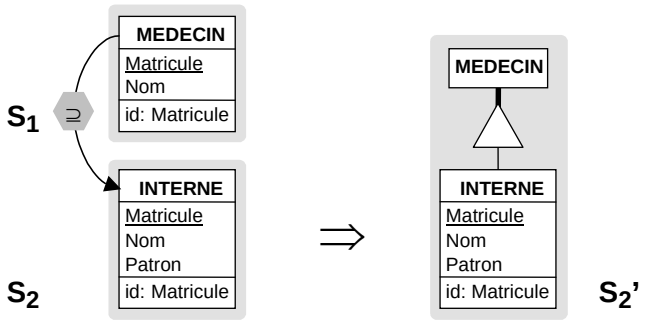


Figure A18.53 - Unification selon une correspondance $\supseteq$ homogène (S_1 inchangé)

d) Correspondance d'union (figure A18.54)

Un type d'entités commun, sans composants, est introduit dans les deux schémas et est déclaré surtype de chaque type d'entités en correspondance. On n'effectue pas à ce stade de migration de composants communs des sous-types vers le surtype. Les types d'associations sont traités de la même manière mais soumis à des contraintes d'inclusion.

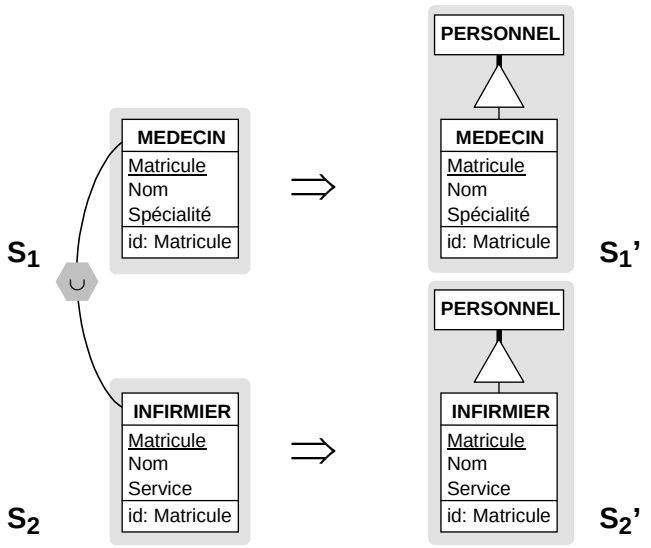


Figure A18.54 - Unification selon une correspondance $\cup$ homogène

e) Correspondance de dérivation

Cette correspondance recouvre une très grande variété de cas de **redondance**, dont les plus représentatifs ont été discutés et traités lors de l'étude de la normali-

sation (en particulier A18.2.6 et A18.2.7). Rappelons quelques exemples : relation *is-a* transitive, attributs copiés ou calculables, types d'associations composés ou projetés. Le traitement consiste à supprimer l'objet (ou groupe d'objets) mineur dans son schéma. L'objet à supprimer est cependant conservé s'il est un composant indispensable dans une autre construction (une contrainte par exemple). Dans ce cas, une contrainte de redondance est ajoutée.

f) Correspondance de dérivation mutuelle

On ramène ce cas à celui de la dérivation simple de la manière suivante. On sélectionne l'un des objets et on le déclare majeur. On réduit ensuite chacun des objets au minimum d'information nécessaire pour recalculer les composants dérivables. On trouvera dans l'étude de cas ci-après un exemple de traitement d'une correspondance de dérivation mutuelle (type d'associations acheté dans le schéma 3).

g) Correspondance de différence

On modifie le nom d'un des deux objets afin de supprimer l'homonymie.

B. Traitement des correspondances hétérogènes positives

Une correspondance hétérogène positive est établie entre deux objets de natures différentes. On la traitera en deux temps : on transformera l'un des objets de manière à ce que ceux-ci soient de même nature, puis on traitera la correspondance homogène ainsi obtenue comme décrit précédemment. Nous analyserons trois couples hétérogènes : attribut/type d'entités, attribut/rôle et type d'associations/type d'entités.

a) Correspondance entre un attribut et un type d'entités (figure A18.55)

L'attribut COMMANDE.Objet est mis en correspondance de compréhension avec le type d'entités ARTICLE. Transformons cet attribut en un type d'entités OBJET par représentation des valeurs. La correspondance $\supseteq$ est à présent définie entre deux types d'entités et devient homogène. Leurs identifiants sont mis en correspondance d'égalité. Après fusion, on observera que OBJET est un sous-type *faiblement spécifique* de ARTICLE. La normalisation proposera sans doute de fusionner ces deux types d'entités (selon figure A18.2.14).

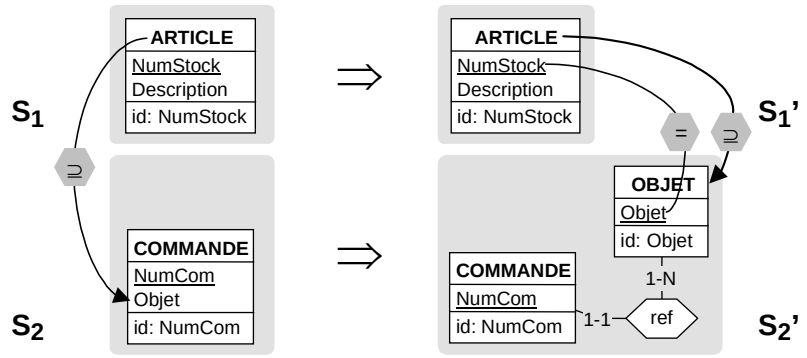


Figure A18.55 - Homogénéisation d'une correspondance *attribut / type d'entités*

b) Correspondance entre un attribut et un rôle (figure A18.56)

L'attribut VEHICULE.vendeur de S_2 est mis en correspondance avec le rôle vendu.SOCIETE dans S_1. Ce pattern est similaire au précédent mais la correspondance est établie avec une variante seulement du type d'entités. On pourrait définir un sous-type VENDEUR dans S_1 et retomber sur le cas précédent, mais il est préférable de laisser inchangé l'objet dominant et de transformer plutôt la construction mineure : l'attribut vendeur. Transformons celui-ci en un type d'entités VENDEUR par représentation des valeurs. SOCIETE et VENDEUR sont mis en correspondance de compréhension mais ceci est insuffisant : il faut encore mettre en correspondance vendu dans S1 avec le nouveau type d'associations. Nous appelons donc ce dernier vendu, ce qui établit une correspondance d'égalité implicite entre eux. Lors de la fusion, VENDEUR sera sans doute considéré comme un sous-type faiblement spécifique et sera supprimé.

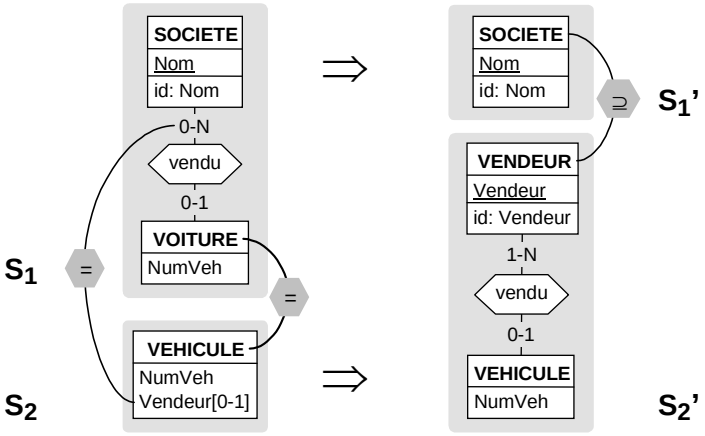


Figure A18.56 - Homogénéisation d'une correspondance *attribut / rôle*

c) Correspondance entre un type d'associations et un type d'entités (figure A18.57)

Le type d'associations fabrique du schéma S_1 est en correspondance α (soit = par exemple) avec le type d'entités PRODUCTION du schéma S_2, ce qui donne à ce dernier le statut de *type d'entités association* (on vérifie qu'il en satisfait les pré-conditions selon A18.2.2).

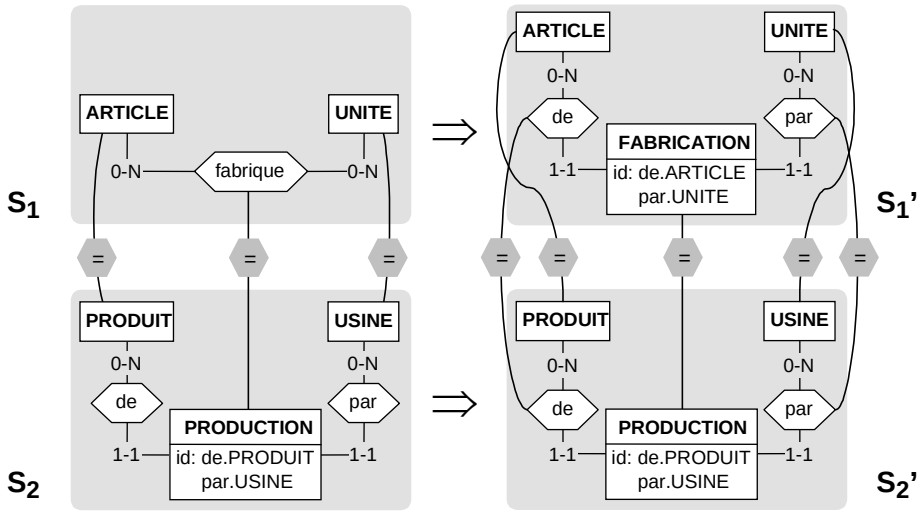


Figure A18.57 - Homogénéisation d'une correspondance *type d'entités / type d'associations*

Les types d'entités ARTICLE et PRODUIT d'une part et UNITE et USINE d'autre part sont en correspondance. Le type d'associations fabrique est transformé en un type d'entités FABRICATION et deux types d'associations fonctionnels. La correspondance = est établie entre FABRICATION et PRODUCTION. Des correspondances sont établies entre les nouveaux types d'associations et ceux du schéma S_2.

A18.4.5 Fusion des schémas unifiés

Grâce à l'unification, les objets en correspondance positive sont de même nature et portent le même nom lorsqu'ils sont identiques. Les seules correspondances qui subsistent sont les correspondances d'*égalité* implicites.

Selon l'ordre des schémas défini en A18.4.1, S_1 est le schéma *maître*, dont l'enrichissement à l'aide des constructions de S_2 produira le résultat S_{12}. La fusion consiste donc à ajouter à S_1 les constructions de S_2 qui lui manquent[27].

On observe cependant que la fusion peut créer des constructions incorrectes ou non normalisées, notamment dans les hiérarchies *is-a* (des composants doivent encore migrer des sous-types vers le surtype ou être supprimés des sous-types, des

27. Formellement : $S_{12} = S_1$; $S_{12} = S_{12} \cup S_2$.

composants redondants doivent être éliminés). On appliquera donc les techniques de correction et de normalisation décrites en A18.1 et A18.2.

A18.4.6 Résolution des conflits sémantiques

Les phases de recherche des correspondances et d'unification ont pour effet secondaire l'identification et la résolution de certains conflits : conflits de *nom* (synonymie, homonymie, erreurs typographiques ou orthographiques), conflits de *modélisation* (les mêmes types de faits sont représentés par des constructions différentes mais équivalentes), conflits de *perspective* (vues plus étroites ou plus générales des mêmes types de faits). Il existe cependant des contradictions entre deux schémas qu'il n'est pas possible de réduire par de simples manipulations de schémas comme nous l'avons fait jusqu'ici : il s'agit des conflits *sémantiques*. Ceux-ci exigent de **corriger** un ou les deux sous-schémas conceptuels à intégrer.

Une première source de conflits sémantiques résulte d'un excès de divergence de *perspective*. Alors qu'on peut sans problème réconcilier les cardinalités [0-5] et [0-10] d'attributs en correspondance d'égalité en [0-10], il faut être plus prudent lorsque le conflit porte sur le caractère *obligatoire ou facultatif*, ou sur le caractère *monovalué ou multivalué*. De même, des types d'associations 1:N et N:1 en correspondance ne se traduisent pas en N:N ou 1:1 sans validation de la part des utilisateurs.

Une seconde source de conflits sémantiques, plus grave, est celle des constructions incorrectes (section A18.1) : la plupart de celles-ci résultent de l'union de deux constructions en elles-mêmes non problématiques. La figure A18.4.58 montre quatre exemples de conflits sémantiques qui interdisent l'intégration des schémas. Ces exemples sont tirés de la section A18.1. Le lecteur en imaginera aisément d'autres.

Quand convient-il de résoudre les conflits sémantiques ? La littérature suggère de les traiter soit lors de l'unification, soit lors de la fusion proprement dite.

A18.4.7 Compléments

L'étude du processus d'intégration que nous avons présentée dans cette section est évidemment loin d'épuiser la question. Signalons quatre prolongements utiles.

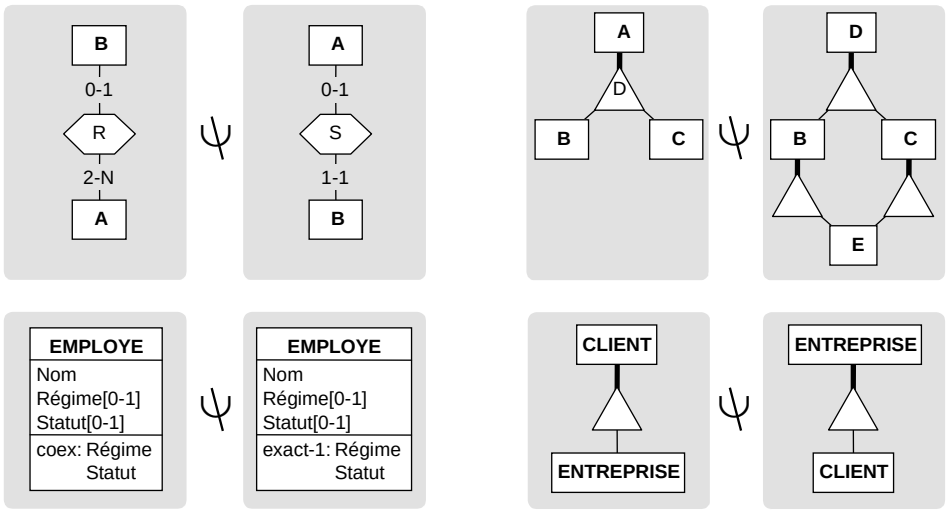


Figure 4.58 - Quatre exemples de conflits sémantiques : schémas non intégrables

- *Intégration de schémas à faible expressivité*. La richesse du modèle Entité-association implique naturellement une plus grande complexité des règles d'intégration. On pourrait dès lors suggérer de transformer au préalable les schémas à intégrer dans un modèle plus pauvre et donc moins expressif (celui du chapitre 11 par exemple), de procéder à leur intégration puis de normaliser le résultat de manière à reconstituer les constructions expressives à l'aide des techniques étudiées en A18.2.2. De nombreux travaux actuels, qui utilisent des structures de données intrinsèquement *arborescentes*, telles qu'XML ou les ontologies, s'inscrivent (inconsciemment) dans cette direction.
- *Auto-intégration*. L'intégration de deux schémas réduit la redondance des structures entre ceux-ci. Rien n'interdit de procéder à l'intégration de deux parties distinctes d'un même schéma de grande taille. Les techniques étudiées dans cette section sont donc applicables à l'élimination des redondances dans le processus de normalisation.
- *Analyse et intégration*. L'analyse de documents (section 18.2 et plus particulièrement 18.2.4) en vue d'en tirer un schéma conceptuel est intrinsèquement un processus d'intégration. En effet, il s'agit d'enrichir le schéma courant d'une construction élémentaire exprimant la sémantique d'un fragment de texte ou d'un formulaire. Cet enrichissement constitue en soi une intégration du schéma en cours et de la construction élémentaire. Dans ce contexte, les opérations de *restructuration* décrites à la section 12.5.5 ne sont rien d'autre qu'une version simplifiée du processus d'unification.
- *Intégration de bases de données*. Nous avons étudié le problème de l'intégration de schémas relatifs à une base de données en projet, c'est-à-dire auxquels ne

correspondent pas encore d'instances. On peut également intégrer des schémas conceptuels de bases de données existantes, obtenus par exemple par rétro-ingénierie. L'existence de données change la problématique de manière importante. D'une part, les hypothèses sur la nature des correspondances peuvent être validées par une analyse des données. Par exemple, la comparaison des populations de deux objets des schémas permet de décider si une correspondance est d'égalité, de compréhension, d'union ou de différence. D'autre part, le but de l'intégration de bases de données est en général d'intégrer non seulement les schémas mais aussi les données. On est alors confronté à des problèmes nouveaux, liés à la qualité des données existantes.

A18.4.8 Étude de cas

On se propose d'intégrer les trois schémas de la figure A18.4.59. Le Schéma 2 semble le plus riche et le plus expressif. Nous le choisirons comme schéma de départ. Les schémas seront pris en compte dans l'ordre Schéma 2, Schéma 1, Schéma 3.

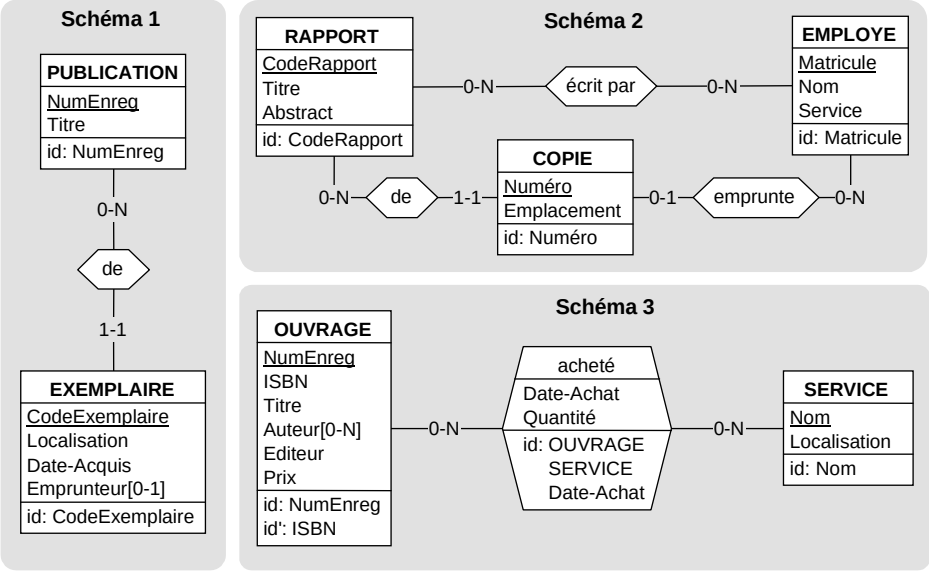


Figure 4.59 - Trois schémas à intégrer

L'analyse comparative des schémas 2 et 1 fait apparaître les correspondances suivantes (figure A18.60), validées par les utilisateurs :

1. PUBLICATION ⊇ RAPPORT ; un *rapport* est un cas particulier de *publication*.
2. EXEMPLAIRE = COPIE ; termes synonymes.
3. de(PUBLICATION, EXEMPLAIRE) ⊇ de(RAPPORT, COPIE ; conséquence de (1) et (2).
4. EXEMPLAIRE.CodeExemplaire = COPIE.Numéro ; termes synonymes attachés

à des objets synonymes.

5. EXEMPLAIRE.Localisation = COPIE.Emplacement ; termes synonymes attachés à des objets synonymes.
6. EXEMPLAIRE.Emprunteur = emprunte.EMPLOYE ; une valeur d'Emprunteur représente une instance d'EMPLOYE jouant un rôle dans emprunte.

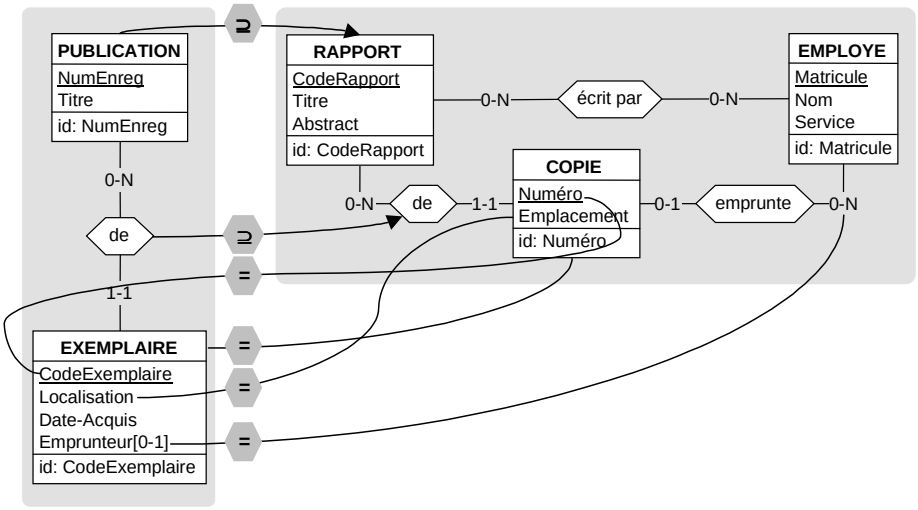


Figure A18.60 - Correspondances entre les schémas 1 et 2

L'unification va modifier les deux schémas de manière à rendre compatibles les objets en correspondance (figure A18.61).

1. Un surtype PUBLICATION de RAPPORT est introduit dans le schéma 2.
2. COPIE est renommé EXEMPLAIRE dans le schéma 2.
3. La correspondance de compréhension, et non d'égalité, entre les types d'associations de est due au fait que RAPPORT est un sous-type de PUBLICATION ; on les considère donc comme égales ; pas d'action donc.
4. Numéro est renommé CodeExemplaire dans le schéma 2.
5. Emplacement est renommé Localisation dans le schéma 2.
6. EXEMPLAIRE.Emprunteur est transformé en un type d'entités EMPLOYE et un type d'associations emprunte ; on donne à l'attribut identifiant d'EMPLOYE du schéma 1 le nom de l'identifiant primaire d'EMPLOYE dans le schéma 2 (Matricule).

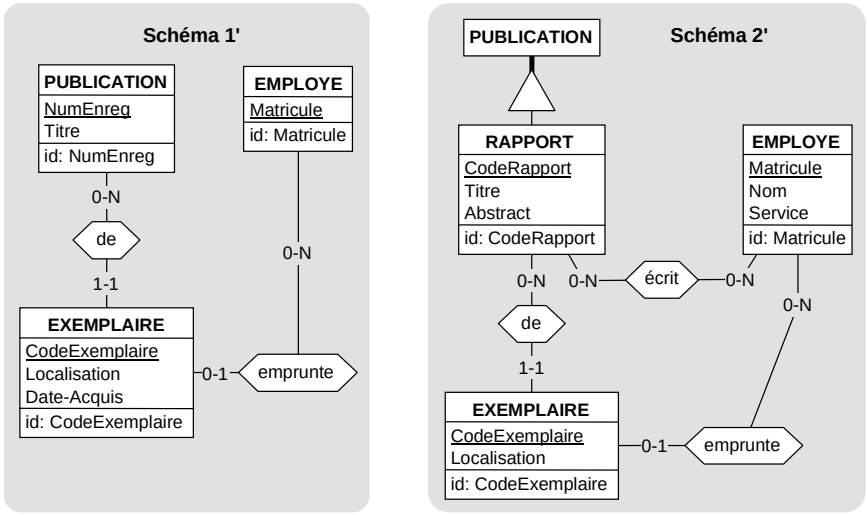


Figure A18.61 - Unification des schémas 1 et 2 en 1' et 2'

La fusion consiste à enrichir le schéma 2' des objets du schéma 1' qu'il ne possède pas :

- les attributs NumEnreg et Titre de PUBLICATION ;
- l'attribut Date-Acquis d'EXEMPLAIRE ;
- le type d'associations de entre PUBLICATION et EXEMPLAIRE.

L'observation de la structure d'héritage nous permet de simplifier le résultat : l'attribut Titre et le rôle de.RAPPORT de RAPPORT sont hérités de PUBLICATION, et peuvent donc être supprimés. L'identifiant primaire propre de RAPPORT entre en conflit avec celui de PUBLICATION. Il devient secondaire (figure A18.62).

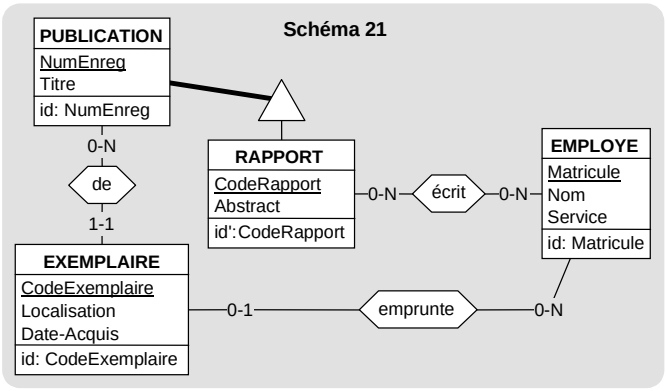


Figure A18.62 - Le schéma 21, résultant de l'intégration des schémas 1' et 2'

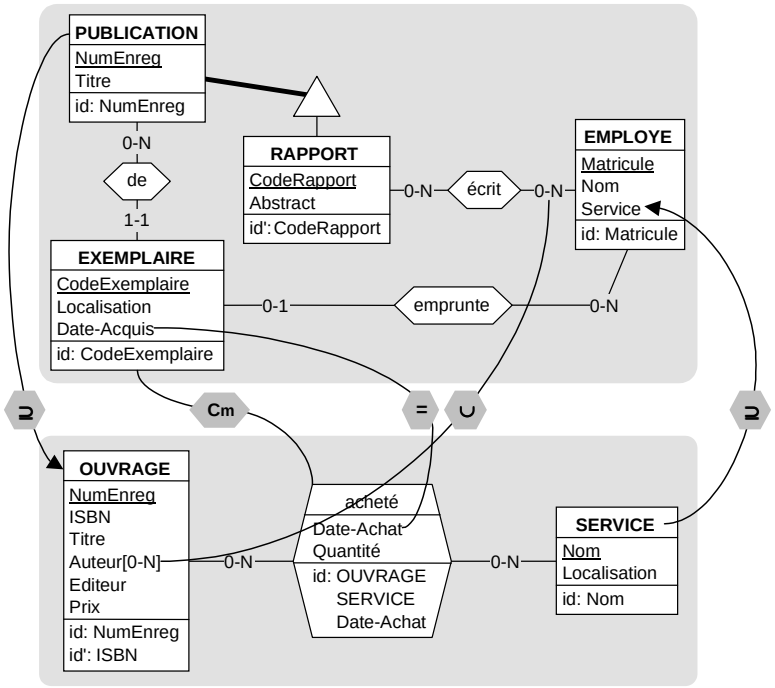


Figure A18.63 - Correspondances entre les schémas 21 et 3

L'intégration des schémas 21 et 3 va introduire un problème nouveau, celui d'une dérivation mutuelle. On relève cinq correspondances (figure A18.63) :

1. PUBLICATION ⊇ OUVRAGE ; OUVRAGE est *une sorte de* PUBLICATION.
2. OUVRAGE.Auteur ∪ écrit.EMPLOYE ; OUVRAGE.Auteur représente une information *similaire* à écrit.EMPLOYE (on rappelle que OUVRAGE et RAPPORT sont deux sortes de PUBLICATION).
3. EXEMPLAIRE Cm acheté; les concepts sous-jacents sont fortement liés sans qu'on puisse établir une correspondance d'égalité, de compréhension, d'union ou de dérivation.
4. acheté.Date-Achat = EXEMPLAIRE.Date-Acquis ; il s'agit d'une correspondance hétérogène en raison de la nature de la correspondance entre leurs parents.
5. SERVICE ⊇ EMPLOYE.Service.

L'unification des schémas 21 et 3 s'effectue comme suit (figure A18.65) :

1. Un surtype PUBLICATION d'OUVRAGE est introduit dans le schéma 3.
2. L'attribut OUVRAGE.Auteur est d'abord transformé en un type d'entités AUTEUR, identifié par l'attribut Nom ; les types d'entités AUTEUR et EMPLOYE héritent de la correspondance d'union. Par conséquent, un surtype PERSONNE d'EMPLOYE est introduit dans le schéma 21 et un surtype PER-

SONNE d'AUTEUR est introduit dans le schéma 3.

3. La correspondance Cm étant hétérogène, nous transformons d'abord acheté en type d'entités ACHAT pour y voir plus clair (A18.64).

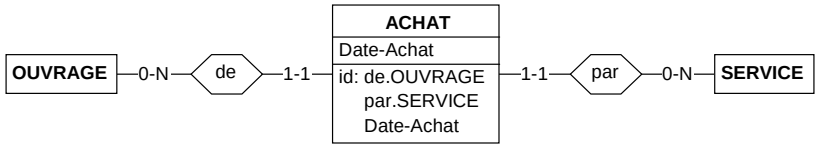
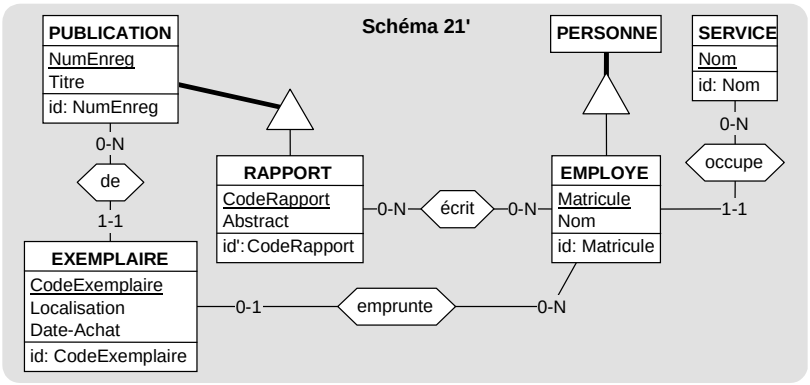


Figure A18.64 - Première tentative d'unification de *acheté* et *EXEMPLAIRE*

Une entité ACHAT apparaît comme une **agrégation** d'entités EXEMPLAIRE : l'ensemble des *exemplaires* d'un *ouvrage* achetés à une certaine *date* par un certain *service* est représenté par une entité ACHAT. ACHAT est donc dérivable du schéma 21 *pour autant qu'on connaisse pour chaque exemplaire le service qui l'a acheté*. On propose de ne conserver du type d'associations acheté d'origine que le type d'associations fonctionnel d'EXEMPLAIRE vers SERVICE : *un exemplaire a été acheté par un service*. On lui conserve le nom acheté.

Dans le schéma 3, on supprime le type d'associations acheté, on introduit un type d'entités EXEMPLAIRE associé à OUVRAGE et on crée un type d'associations N:1 acheté entre EXEMPLAIRE et SERVICE.



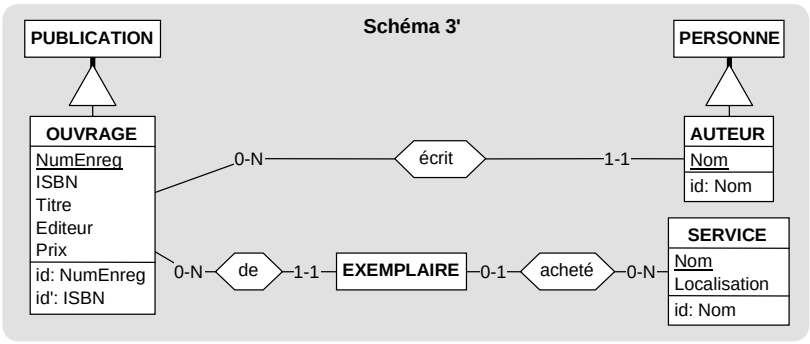


Figure A18.65 - Unification des schémas 21 et 3 en 21' et 3'

4. L'attribut EXEMPLAIRE.Date-Acquis est renommé Date-Achat.
5. On introduit dans le schéma 21 un type d'entités SERVICE et un type d'associations occupe entre SERVICE et EMPLOYE. SERVICE reprend l'identifiant de son équivalent dans le schéma 3.

La figure A18.66 montre le résultat de la fusion des schémas 21' et 3' unifiés. Les caractéristiques communes des sous-types ont été remontées dans le surtype. Il reste quelques questions à régler, notamment les contraintes de sous-types. On propose la *partition* pour PUBLICATION et la *totalité* pour PERSONNE. Cependant, le fait que AUTEUR et EMPLOYE possèdent chacun un identifiant primaire pose problème : certaines entités PERSONNE vont avoir *deux identifiants primaires*. On change donc le statut de l'identifiant de AUTEUR de primaire en secondaire.

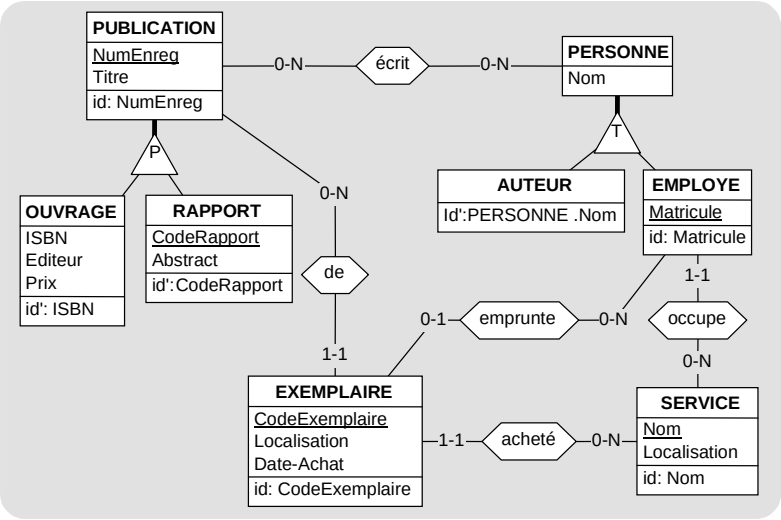


Figure A18.66 - Le schéma global intégré

A18.5EXERCICES DU CHAPITRE 18

A18.1 *Toute personne possède deux parents biologiques, lesquels sont des personnes*. Discuter cette proposition dans le contexte de l'analyse conceptuelle.

Solution

Introduire dans le schéma, comme une analyse naïve pourrait le suggérer, un rôle enfant de cardinalité [2-2] rendrait le type d'entités PERSONNE **non satisfiable** (section 17.6.2/c). On se souviendra qu'un schéma conceptuel ne représente pas à proprement parler un domaine d'application, mais plutôt la connaissance qu'on en a. Comme on ne connaît pas les parents biologiques de chaque personne, on dégradera la contrainte en [0-2].

A18.2 Proposer un schéma conceptuel explicitant les relations existant parmi les *personnes* formant des *familles* et/ou des *ménages*. On veillera à représenter l'évolution de ces structures.

A18.3 Construire le schéma conceptuel décrivant un réseau de distribution d'eau.

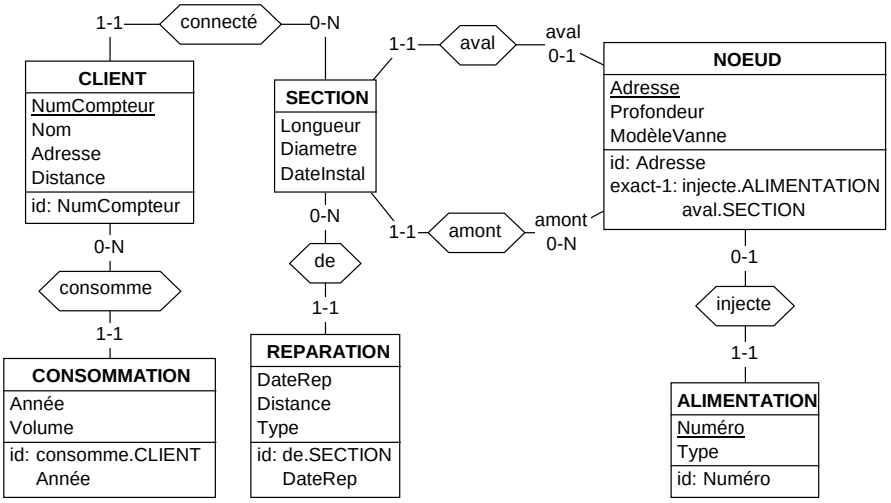
Un réseau de distribution d'eau est constitué de sections de canalisation. Une section relie deux nœuds : le nœud amont (alimentant) et le nœud aval (alimenté); l'eau s'écoule donc dans un sens déterminé, de l'amont vers l'aval. Une section est caractérisée par sa longueur, son diamètre, sa date d'installation et la liste des réparations qu'elle a subies (date, distance par rapport au nœud amont, type). A l'exception des nœuds terminaux ("racines" et "feuilles"), un nœud relie une section amont (alimentant le nœud) et une ou plusieurs sections aval (alimentées par le nœud), l'ensemble formant une forêt (ensemble d'arbres).

Un nœud comprend une vanne dont on connaît le modèle. La racine d'un arbre est un nœud spécial qui n'est pas connecté à une section amont, mais qui reçoit l'eau d'une installation d'alimentation, identifiée par un numéro, et d'un type tel que "captage", "réservoir", "station d'épuration", etc. Chaque installation d'alimentation alimente un arbre du réseau. Les feuilles d'un arbre n'alimentent pas de sections aval. Chaque nœud est repéré par son adresse; il est caractérisé par sa profondeur. Une section ne fait pas l'objet de plus d'une réparation par jour.

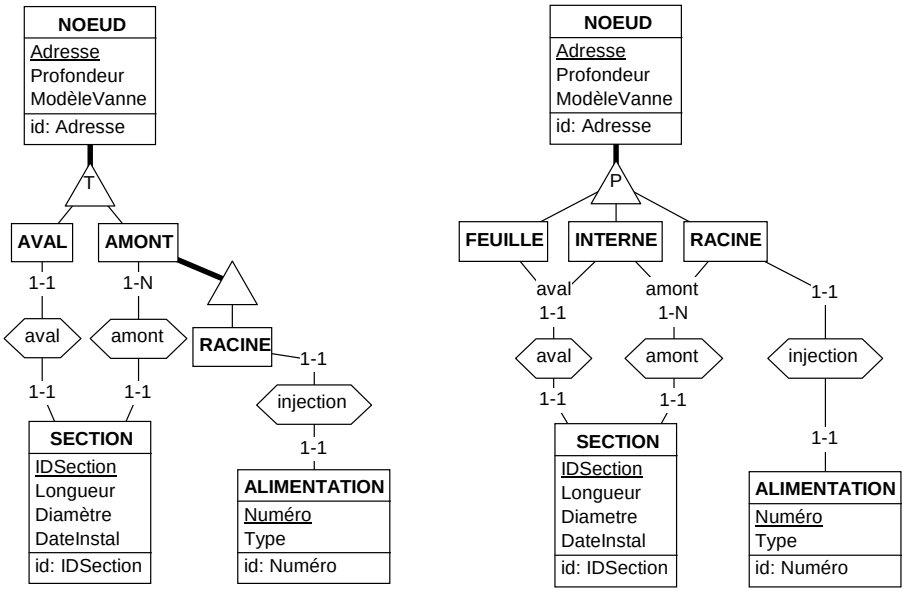
Chaque client (nom, adresse) possède un et un seul compteur identifié par son numéro, et branché sur une section. La position de ce branchement est indiquée par la distance à partir du nœud amont. Pour chaque client, on enregistre les consommations annuelles.

Solution

Première approche simplifiée :



Remarques. (1) SECTION, un concept majeur, ne dispose que d'un identifiant implicite (son NOEUD aval), (2) les différentes sortes de noeuds ne sont pas mises en évidence. D'où les deux variantes ci-dessous. Certaines contraintes sont manquantes.



A18.4 Etudier soigneusement les principales pages d'un site de commerce électronique tel que www.amazon.fr, www.fnac.com ou www.ebay.fr. Il apparaît clairement que les informations qu'elles contiennent sont extraites

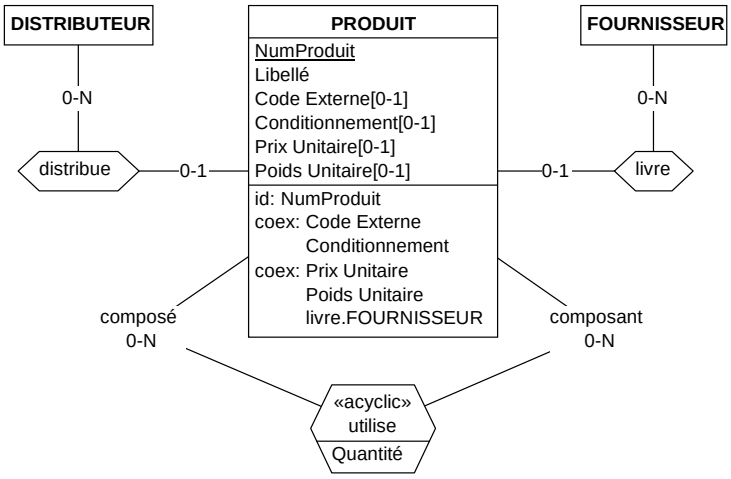
d'une base de données. Identifier dans le site choisi une partie réduite, telle que *Musique / Meilleures ventes* chez Amazon, et proposer un schéma conceptuel des concepts et des informations qui y sont représentés.

A18.5 Reprenons le problème consistant à représenter un ensemble de produits et leur composition, déjà rencontré à plusieurs reprises dans cet ouvrage :

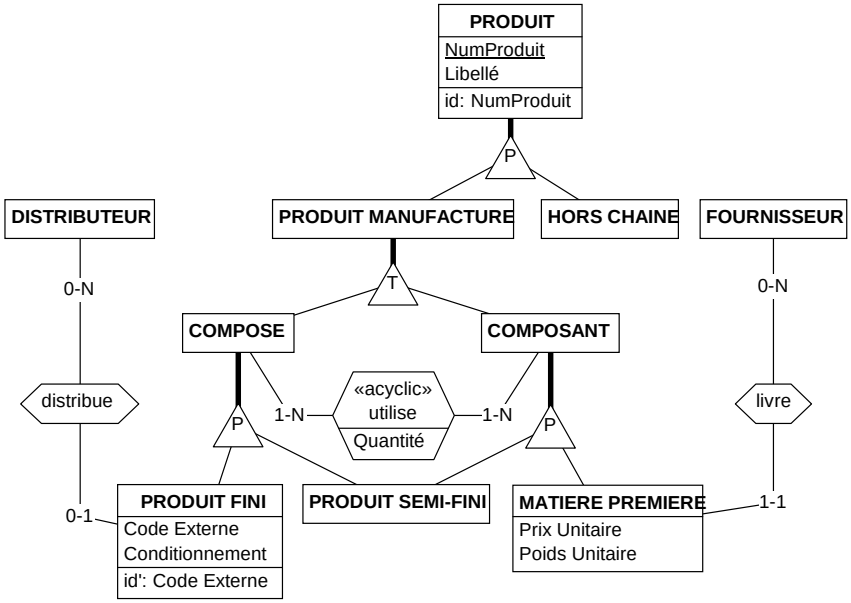
Chaque produit peut être composé de produits plus simples, eux-mêmes éventuellement composés d'autres produits. Pour chaque produit composé, on indique la quantité de chaque composant qui est nécessaire pour en fabriquer une unité. Pour les produits élémentaires (non composés) on connaît le prix unitaire, le poids unitaire et le fournisseur qui peut le livrer. Tout produit a un numéro identifiant et un libellé. Un produit fini possède un code externe identifiant, un conditionnement et éventuellement un distributeur. Un produit n'intervient pas, même indirectement, dans sa propre composition.

Solution

L'énoncé indique que la structure des produits est un *graphe hiérarchique*. Nous proposons d'abord un schéma très simple



Il peut cependant être utile de distinguer les différentes classes de produits, qui sont dotées de propriétés spécifiques. L'énoncé a déjà identifié les produits élémentaires, communément dénommés *matières premières* et les produits finis. Le schéma ci-dessous est une spécialisation du précédent. Il détaille les autres classes de produits et assigne à certaines d'entre elles des propriétés spécifiques, sous la forme d'attributs, de rôles et d'identifiants.



PRODUIT FINI $\cap$ MATIERE PREMIERE = $\varnothing$

A18.6 La partie *achat* du site commercial mentionné à la question précédente comporte aussi des informations relatives aux clients, aux paiements et aux conditions de livraison. Il est évident que la société conserve des informations sur les achats précédents des clients, de manière à personnaliser les pages qui sont présentées à un visiteur particulier.

Proposer une extension du schéma de la question précédente qui décrive ces nouvelles informations.

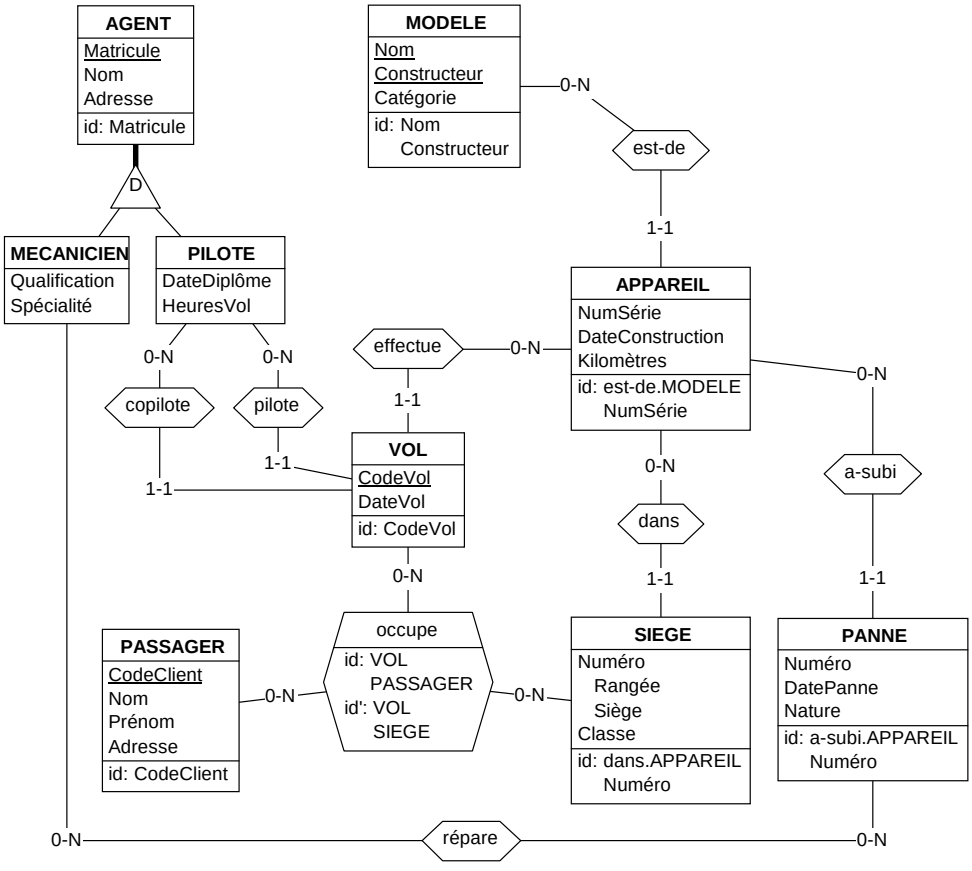
Remarque. Les exercices A18.4 et A18.6 illustrent une problématique qui prend actuellement de plus en plus d'importance : comprendre la structure sémantique de sites étrangers afin d'en extraire de manière intelligente des données pertinentes. Cet exercice relève de la *rétro-ingénierie des site web*.

A18.7 Proposer un schéma conceptuel traduisant les concepts de l'énoncé suivant, relatif aux activités d'une compagnie aérienne.

Des appareils effectuent des vols, chacun identifié par un code de vol et caractérisé par la date à laquelle il a été effectué. Chaque appareil appartient à un certain modèle, lui-même caractérisé par son nom, son constructeur et la catégorie d'appareil. Les modèles d'appareils se distinguent par leur constructeur et leur nom. Chaque appareil a un numéro de série (qui le distingue des autres appareils de son modèle), une date de construction et un kilométrage. Un appareil peut avoir subi des pannes. Chacune d'elles est caractérisée par un numéro (unique parmi les pannes de l'appareil), sa date et sa nature. Les pannes sont réparées par un certain

nombre de mécaniciens, qui sont des employés, identifiés par leur matricule, et caractérisés par leur nom et leur adresse. On connaît aussi la qualification et la spécialité des mécaniciens. Un appareil possède des sièges numérotés (p. ex. 12A = 12e rangée, 1er siège) caractérisés par leur classe. Lors d'un vol, chaque siège de l'appareil qui l'effectue peut être occupé par un passager (nom, prénom, adresse, code client identifiant); lors d'un vol, un passager n'occupe qu'un seul siège. Durant un vol, l'appareil est pris en charge par un pilote et un copilote. Ceux-ci sont des employés dont on connaît en outre la date du diplôme et le nombre d'heures de vol.

Solution



A18.8 Proposer un schéma conceptuel traduisant les concepts de l'énoncé suivant.

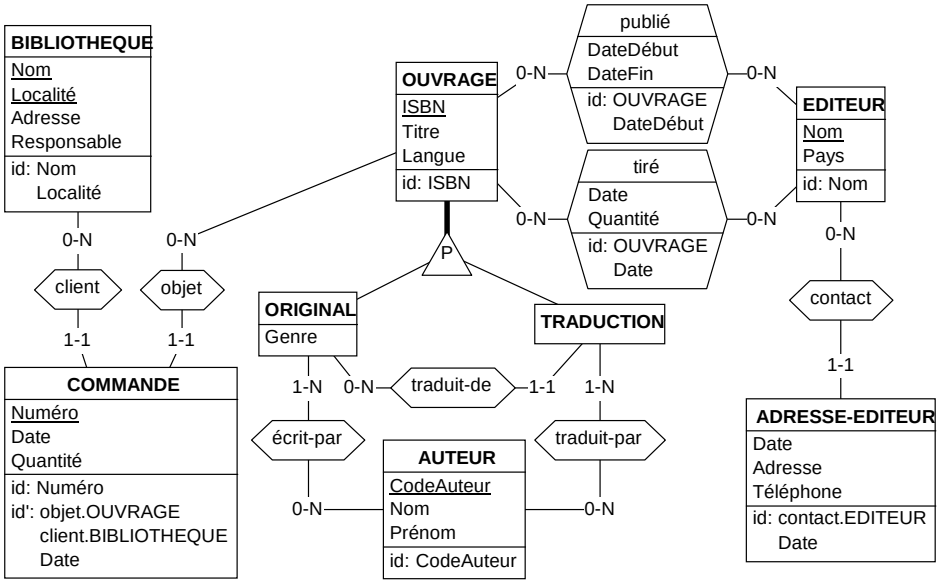
Dans un réseau de bibliothèques, chacune d'elles commande des exemplaires d'ouvrages repris dans une liste de référence. Chaque commande est numérotée (de manière à être identifiable) et reprend la quantité, la date et

l'ouvrage concerné. Une bibliothèque n'envoie pas plus d'une commande d'un même ouvrage le même jour.

Un ouvrage peut être publié par un éditeur pendant une certaine période. A tout moment, l'éditeur d'un ouvrage peut céder ses droits de publication à un autre éditeur (il peut même les réacquérir plus tard). D'un ouvrage, un éditeur qui en possède les droits peut tirer, à une certaine date, un nombre déterminé d'exemplaires. Si, après au moins 3 mois, les ventes s'avèrent meilleures que prévu, l'éditeur effectuera un nouveau tirage (et ainsi de suite). Un ouvrage est caractérisé par un numéro ISBN qui l'identifie, son titre et sa langue. Un ouvrage est soit un original (auquel cas il est caractérisé par un genre), écrit par un ou plusieurs auteurs, soit une traduction d'un original. Dans ce dernier cas, l'ouvrage est traduit par un ou plusieurs auteurs (pour simplifier, vrais auteurs et traducteurs sont considérés comme des auteurs). Une bibliothèque est identifiée par son nom et sa localité. Elle a une adresse et un responsable.

Un éditeur est identifié par son nom et caractérisé par son adresse et son pays d'origine. Cependant, son adresse peut changer au cours du temps. L'adresse est accompagnée d'un numéro de téléphone. Un auteur porte un code unique et est caractérisé par son nom et son prénom.

Solution



A18.9 Un organisme gère les activités hôtelière d'un ensemble d'établissements d'une région. Chaque hôtel porte un nom unique, appartient à une catégorie, et est ouvert durant une période déterminée. On enregistre son adresse. Il dispose de chambres, classées par catégories, propres à l'hôtel. Le prix d'une

chambre dépend de sa catégorie, qui précise l'équipement disponible : bain, douche et téléviseur. Les chambres d'un hôtel ont des numéros distincts. Elles sont situées à un étage et offrent un certain nombre de lits à une personne et à deux personnes.

Les clients occupent des chambres, avec ou sans réservation. Lors d'une réservation, dont on enregistre le numéro et la date, le client demande une ou plusieurs chambres. Pour chacune, il précise soit une chambre précise soit une catégorie, ainsi que la période d'occupation prévue. L'hôtelier vérifie s'il dispose de chambres de cette catégorie durant cette période. Par la suite, il pourra affecter une chambre réelle à la réservation selon la catégorie.

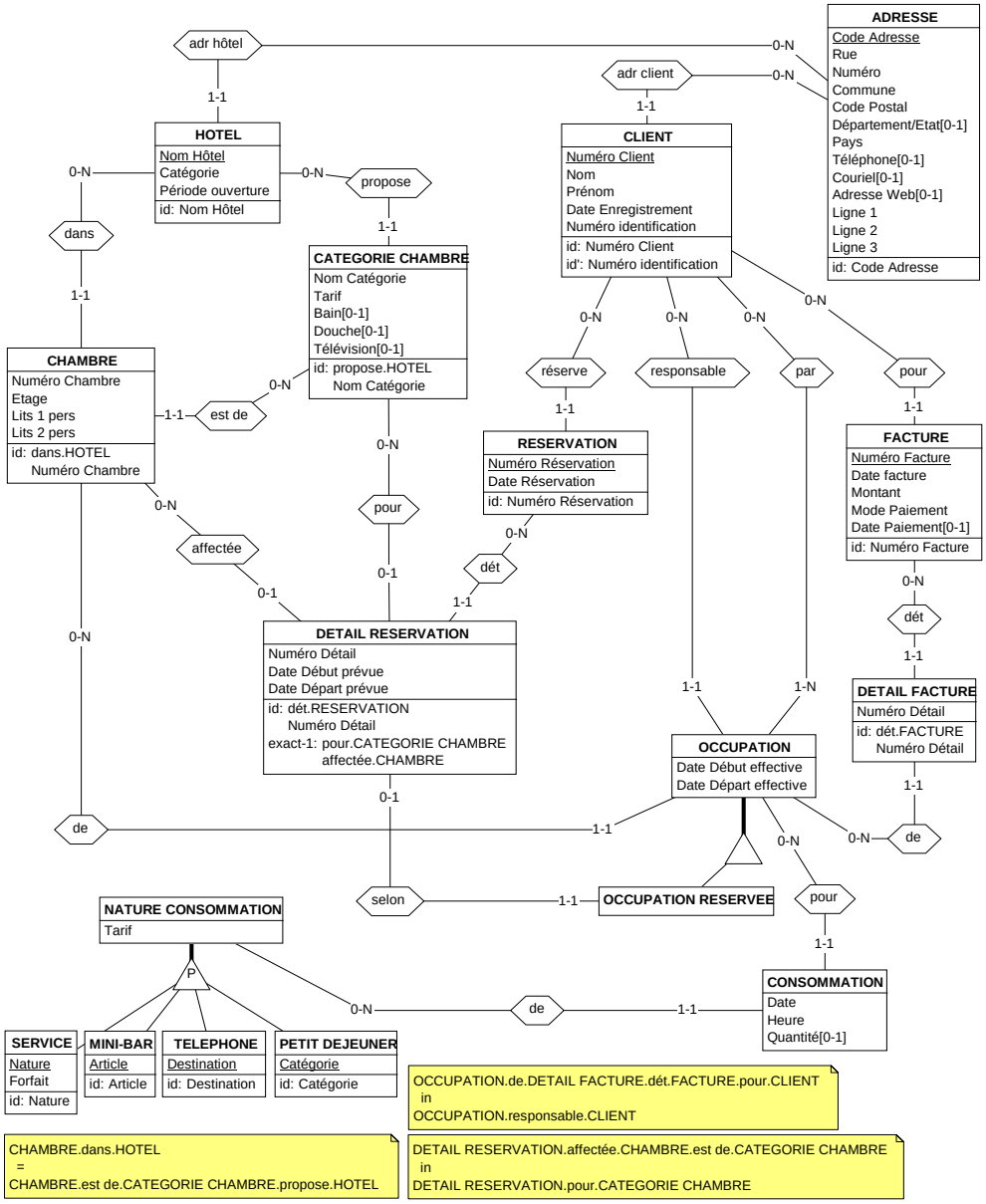
Une chambre est occupée par un ou plusieurs clients durant une certaine période effective. Si cette occupation a fait l'objet d'une réservation, on indique celle-ci.

Les clients occupant une chambre peuvent consommer des services (blanchisserie, sauna, etc.), des articles du mini-bar, des communications téléphoniques (dont le tarif dépend de la destination) et des petits déjeuners (il existe plusieurs catégories, de prix différents). On connaît le tarif de chaque type de consommation; pour un service on indique en outre si le prix est forfaitaire ou proportionnel. Pour chaque consommation, on connaît la date, l'heure et, si pertinent, la quantité. Le responsable de l'occupation d'une chambre est le client qui assumera le paiement; il n'occupe pas nécessairement cette chambre. Lors de l'enregistrement d'un client, on lui affecte un numéro de client et on note son nom, son prénom et son adresse.

Une adresse est composée d'un nom de rue, d'un numéro d'immeuble, du nom de la commune et du code postal, du département ou de l'état (si pertinent), du pays, et, s'ils existent, d'un numéro de téléphone, d'une adresse de courriel et d'une adresse web. On enregistre aussi la même adresse sous la forme d'une à trois lignes qui sont à imprimer telles quelles sur les enveloppes. Plusieurs clients peuvent partager la même adresse.

A la date du départ, une facture est établie et remise au client responsable des occupations. La facture est numérotée et reprend en outre le mode de paiement et, lorsque celui-ci est effectué, sa date. La facture détaille les chambres occupées ainsi que les consommations de chacune. Le montant de la facture peut être différent de la somme des chambres occupées et des consommations (suite à une remise par exemple).

Solution



Remarque

La solution suggérée comporte une contradiction qui rend un type d'entités non satisfiable. Lequel ?

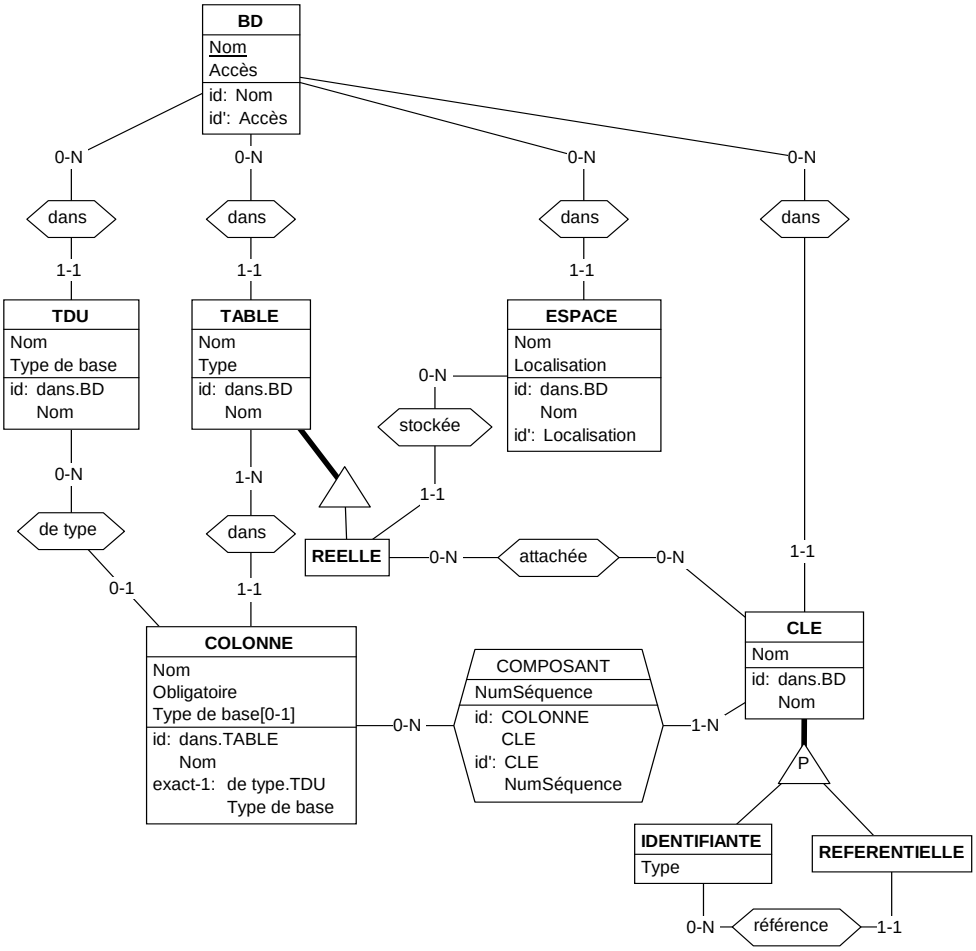
A18.10 Proposer un méta-schéma pour le modèle SQL2 (exprimé dans le modèle Entité-associations étendu). Plus précisément, on désire construire un dictionnaire de données qui représente les structures d'une collection de bases de données. Chaque base de données porte un nom identifiant et est localisée par son chemin d'accès (URL).

Chaque base de données est constituée d'un nombre quelconque de tables, elles-mêmes constituées chacune d'au moins une colonne. Une table est réelle (de base) ou virtuelle (vue). Les lignes d'une table sont stockées dans un espace de stockage de la base de données. Les règles d'unicité des noms des tables, des colonnes et des espaces sont celles d'SQL. La localisation d'un espace de stockage est unique. Une colonne est obligatoire ou facultative. Elle est définie sur un type de base (*decimal*, *character*, *bit*, etc.) ou sur un type défini par l'utilisateur. Ce dernier porte un nom unique dans la base de données et est défini sur un type de base.

Un certain nombre de clés peuvent être attachées à une table de base. Une clé est identifiante (auquel cas elle est primaire ou secondaire) ou référentielle (clé étrangère). Elle porte un nom unique dans la base de données. Une clé est composée d'une séquence d'au moins une colonne de sa table. Une colonne ne peut apparaître plus d'une fois dans une même clé. A chaque clé référentielle est associée une clé identifiante de même composition.

Solution

On propose le schéma ci-dessous. Il devrait être complété de diverses contraintes d'intégrité. On pourrait aussi le compléter par la représentation des *View*, des *index*, des *triggers*, des *checks* et des *privilèges*.



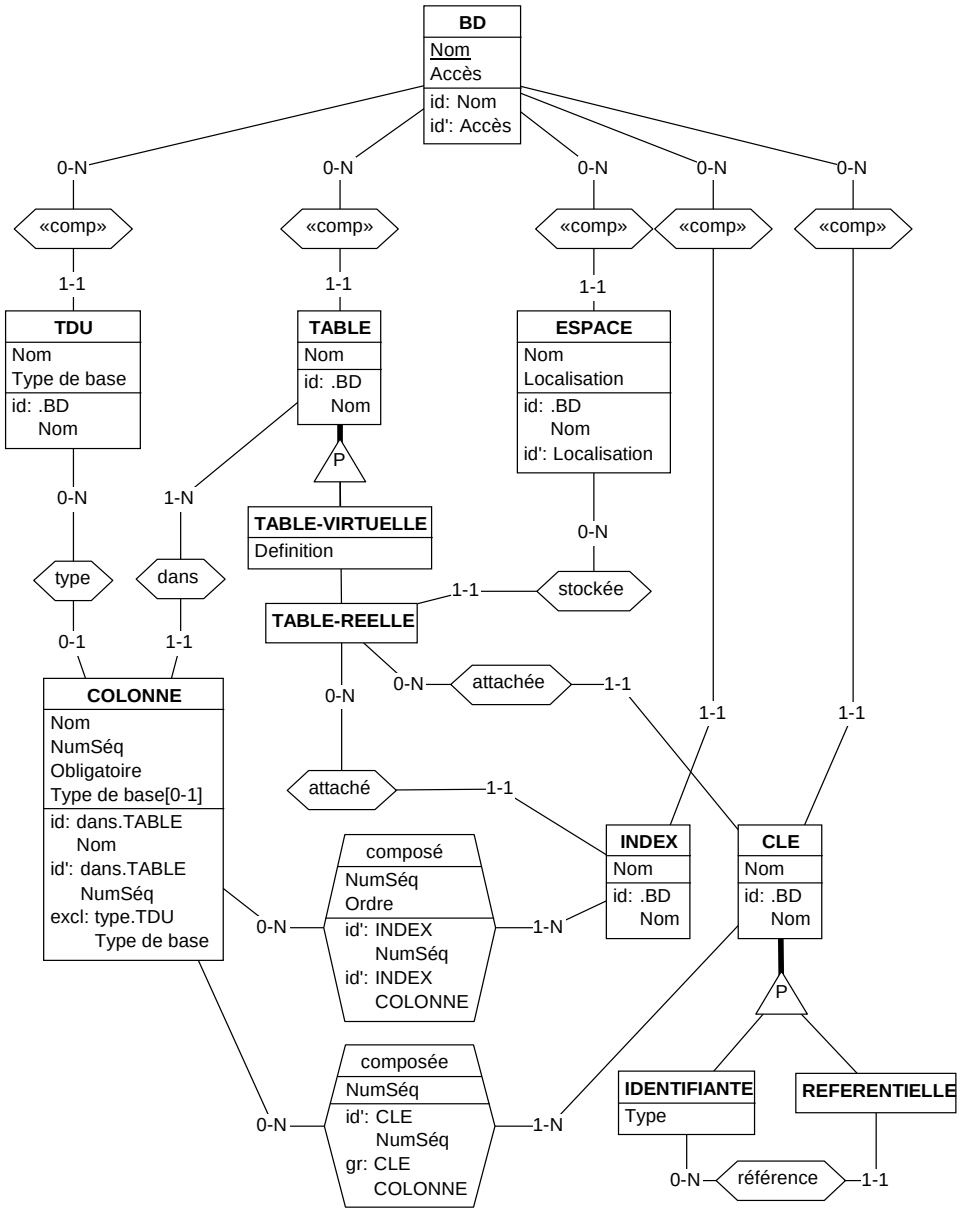


Figure A18.67 - Méta-schéma SQL2. Une version plus complète

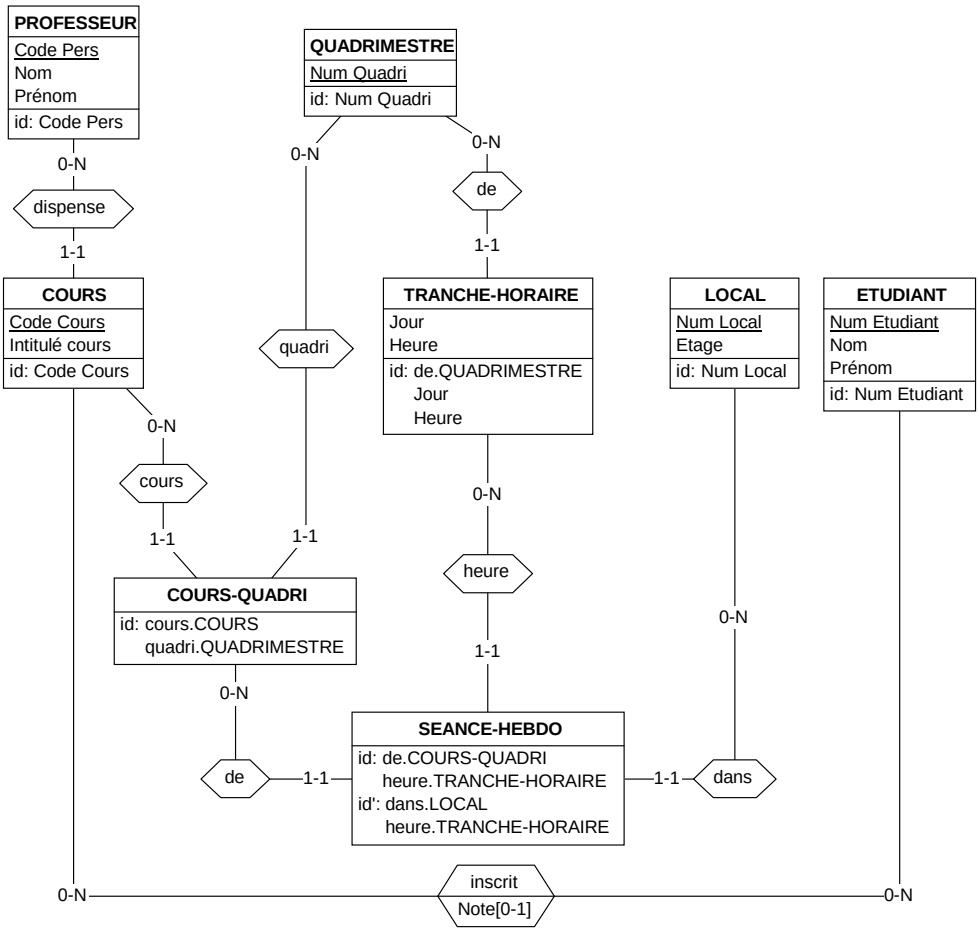
A18.11 Proposer un schéma conceptuel traduisant les concepts de l'énoncé suivant, relatif à l'organisation d'une année académique dans une école.

Chaque cours est dispensé par un professeur et s'étend sur un ou deux quadrimestres complet(s) de l'année académique. Dans chacun de ses

quadrimestres, un cours occupe une ou plusieurs des 20 tranches horaire de 2 heures de la semaine, et pour chacune de ces tranches il est organisé dans un local déterminé. Cette organisation est identique pour toutes les semaines d'un quadrimestre. Les étudiants sont inscrits à des cours, pour chacun desquels ils peuvent obtenir une note.

On veillera à traduire les contraintes suivantes :

1. *pendant une tranche horaire déterminée, un local n'accueille qu'un seul cours,*
2. *pendant une tranche horaire déterminée, un cours ne se déroule que dans un seul local,*
3. *pendant une tranche horaire déterminée, un professeur ne dispense qu'un seul cours,*
4. *pendant une tranche horaire déterminée, un étudiant n'assiste qu'à un seul cours,*
5. *pendant une tranche horaire déterminée, un étudiant ne se trouve que dans un seul local,*
6. *pendant une tranche horaire déterminée, un professeur ne se trouve que dans un seul local.*



"le quadrimestre de la tranche horaire est celui de la partie du cours"
SEANCE-HEBDO.heure.TRANCHE-HORAIREde.QUADRIMESTRE
=
SEANCE-HEBDO.de.COURS-QUADRI.quadri.QUADRIMESTRE

"un professeur ne dispense qu'un seul cours pendant une tranche horaire déterminée"
id(SEANCE-HEBDO):
de.COURS-QUADRI.cours.COURS.dispense.PROFESSEUR,
heure.TRANCHE-HORAIRE

"un étudiant ne se trouve que dans un seul local pendant une tranche horaire déterminée"
id(SEANCE-HEBDO):
de.COURS-QUADRI.cours.COURS.inscrit.ETUDIANT,
dans.LOCAL

"un étudiant n'assiste qu'à un seul cours pendant une tranche horaire déterminée"
id(SEANCE-HEBDO):
de.COURS-QUADRI.cours.COURS.inscrit.ETUDIANT,
heure.TRANCHE-HORAIRE

A18.12 Proposer un schéma conceptuel traduisant les concepts de l'énoncé suivant, relatif à la gestion des clés dans une institution d'enseignement supérieur.

La faculté possède un certain nombre de locaux accessibles par au moins une porte. Chaque local est caractérisé par un nom et un usage. Par exemple, le local I2 est utilisé comme auditoire, le local S2 est une salle de séminaire et le local 309 est utilisé comme bureau. Il n'existe pas deux locaux de même nom. Chaque porte est identifiée par un numéro et est équipée d'une serrure fermant à clé. Certaines portes, telles que les portes d'entrée ainsi que les portes d'étage, ne donnent pas (directement) accès à des locaux. Toutes les clés sont identifiées par un numéro et pour chacune d'elles on connaît le nombre total d'exemplaires, les portes qu'elle ouvre ainsi que le nombre d'exemplaires distribués et la date à laquelle chacun d'eux a été donné. Pour chaque porte on connaît la ou les clés qui permettent de l'ouvrir.

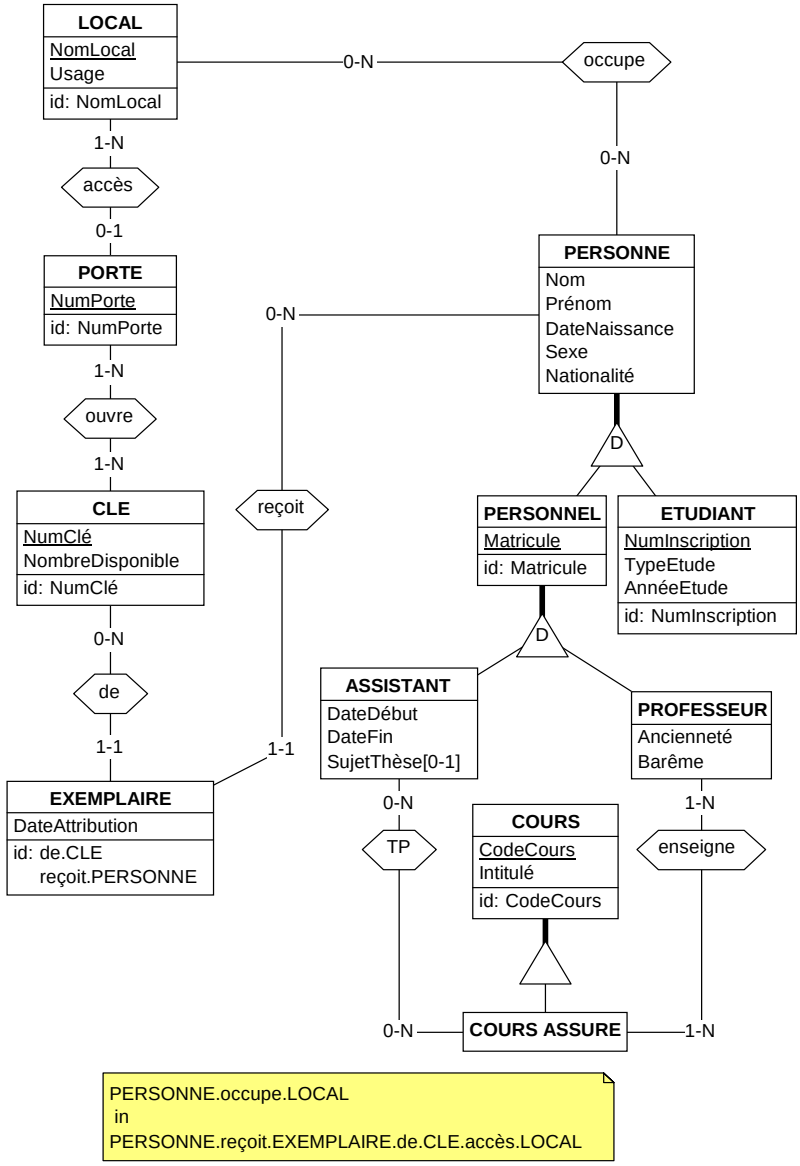
Toutes les personnes (professeur, assistant, étudiant) ayant régulièrement accès à la faculté sont répertoriées. On connaît de chacune d'elles le nom, le prénom, la date de naissance, le sexe, et la nationalité. Si la personne est :

- *un professeur, on connaît en outre ses années d'ancienneté dans la faculté, son échelle barémique, son numéro matricule qui est unique ainsi que les cours qu'il assure.*
- *un assistant, on connaît les dates de début et de fin de son contrat, son éventuel sujet de thèse et les cours dont il assure les travaux pratiques, à l'occasion en collaboration avec d'autres assistants. Chaque assistant a aussi un numéro matricule unique, distinct de ceux des professeurs.*
- *un étudiant, on connaît aussi le type et l'année d'étude en cours ainsi que son numéro d'inscription, qui est unique.*

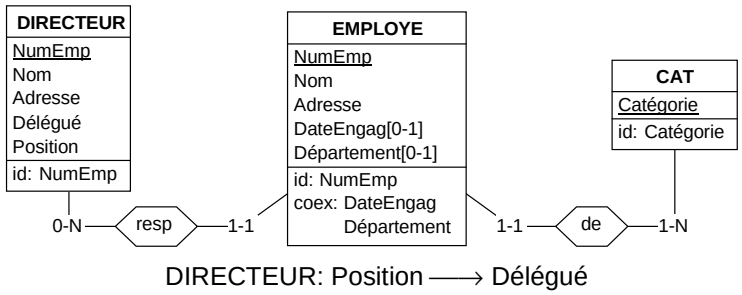
Chaque cours est caractérisé par un intitulé et un code unique. Un même cours peut être assuré par plusieurs professeurs mais un cours peut ne pas être organisé durant l'année courante. Un cours qui fait l'objet de travaux pratiques doit être assuré (pas de cours, pas de TP !).

Un local peut être attribué à une ou plusieurs personnes et certaines personnes peuvent être logées dans un ou plusieurs locaux (par exemple, le doyen possède un bureau de fonction mais aussi un bureau personnel). En fonction des besoins, des clés sont mises à disposition des personnes (sous certaines circonstances un étudiant peut parfois disposer de certaines clés). Une personne ne possède qu'une clé d'un numéro donné mais possède au moins une clé donnant accès à son (ou ses) local(ux).

Solution



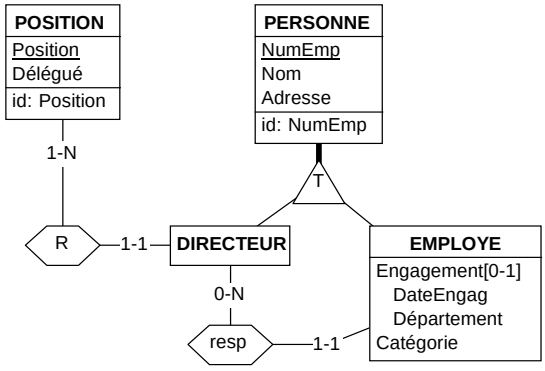
A18.13Normaliser le schéma ci-dessous.



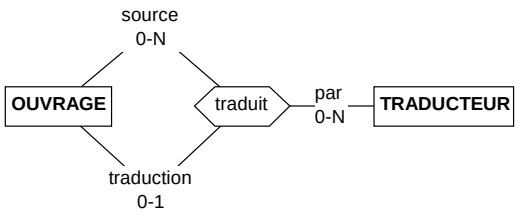
Solution

Ce schéma comporte quatre sous-structures non normalisées.

- DIRECTEUR et EMPLOYE ont des attributs commun. On les extrait pour former le surtype PERSONNE. On fait l'hypothèse que l'identifiant NumEmp s'applique à l'union des populations des sous-types.
- La contrainte coex sur les attributs d'EMPLOYE s'exprime de manière plus claire sous la forme d'un attribut composé facultatif.
- Le type d'entités CAT apparaît comme un *type d'entités - attribut.*
- Le déterminant de la DF explicite Position ⟶ Délégué n'est pas un identifiant de DIRECTEUR. On extrait les attribut litigieux Position et Délégué sous la forme d'un type d'entités autonome.

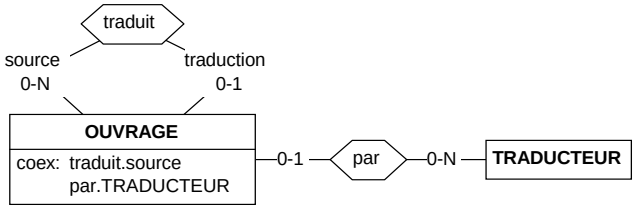


A18.14Normaliser le schéma ci-dessous, qui exprime le fait que certains ouvrages ont été traduits par des traducteurs.



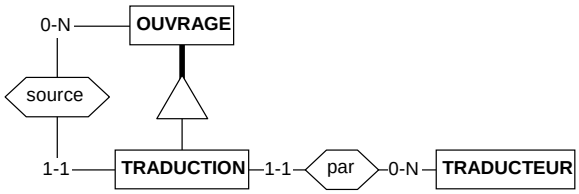
Solution

Ce schéma n'est pas normalisé, car il comporte un type d'associations ternaire dont un rôle est de cardinalité [0-1]. Cette propriété n'entraînant pas de problème de redondance, il est possible de conserver ce schéma tel quel. Si on désire pousser plus loin la normalisation, on décomposera le type d'associations, ce qui conduit au schéma suivant :



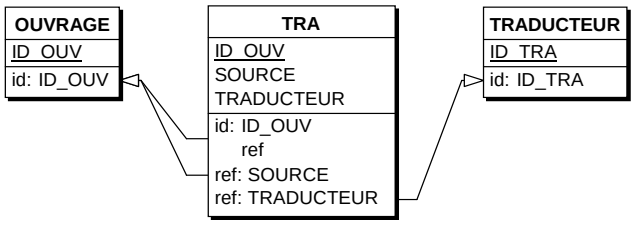
Ce schéma présente deux défauts. D'une part, il introduit une contrainte additionnelle (*coexistence*), et d'autre part il comporte deux rôles facultatifs susceptibles d'entraîner des problèmes dans une implémentation relationnelle (clés étrangères à valeur null).

La contrainte de coexistence peut s'exprimer de manière structurelle via le sous-type TRADUCTION, qui met en évidence le concept d'*ouvrage traduit*. Ce schéma sera sans doute plus évolutif que les précédents si ce concept devait prendre de l'importance dans la suite. En outre, il ne contient plus les rôles facultatifs problématiques.



Discussion additionnelle

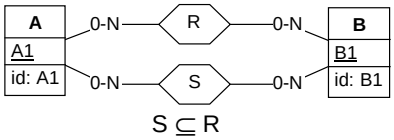
Il est intéressant d'observer que le premier et le troisième schéma se traduisent, selon le plan de transformation classique, par des schémas relationnels apparemment identiques :



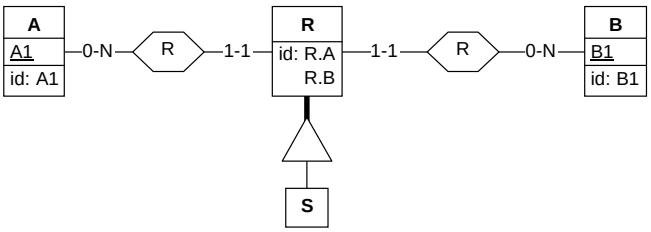
En fait, ces schémas diffèrent par l'interprétation de la table TRA (dont le nom a expressément été choisi pour sa neutralité : TRADUIT ou TRADUCTION). Dans le premier cas, cette table représente les opérations de traduction, les ouvrages traduits étant représentés exclusivement dans la table OUVRAGE. Dans le deuxième cas, la table TRA représente les ouvrages traduits. Un tel ouvrage est représenté par un fragment extrait de TRA auquel on joint le fragment correspondant dans la table OUVRAGE. Si, dans le futur, il était nécessaire d'ajouter des propriétés spécifiques aux ouvrages traduits, il faudrait les ajouter à la table OUVRAGE dans le premier cas, sous forme de composants facultatifs, et à la table TRA dans le second.

Dans le modèle relationnel objet, qui supporte les relations *is-a*, le différence serait bien entendu nettement marquée, et cette discussion serait sans objet.

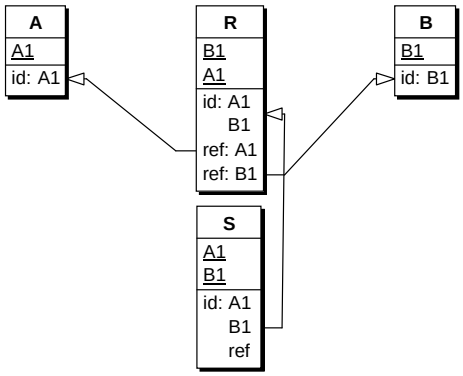
A18.15 Eliminer la contrainte d'inclusion ci-dessous en la représentant de manière structurelle.



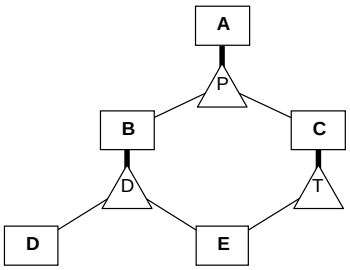
La construction la plus fréquente qui soit basée sur une contrainte d'inclusion d'extensions est la relation *is-a*. En transformant les deux types d'associations en types d'entités, la contrainte d'inclusion se traduit par une relation *is-a* entre ces derniers. On obtient ainsi le schéma ci-dessous.



Il est également possible de simplifier ce schéma selon le procédé de la section 17.7.1 (*Sous-type faiblement spécifique*). On notera que le schéma relationnel qui dérive naturellement de ce schéma normalisé est simple et intuitif :

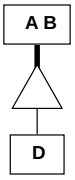


A18.16En utilisant le concept de *satisfiabilité*, simplifier le schéma ci-dessous

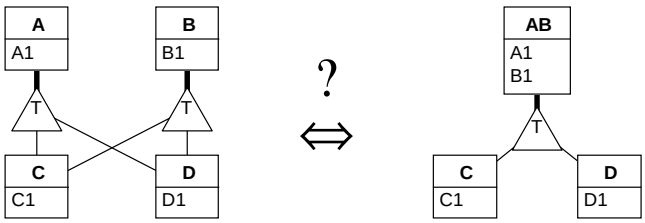


Solution

En raison de la contrainte **P** sur A, le type d'entités E est non satisfiable. Les populations de C et E sont les mêmes (contrainte **T** sur C). C est donc lui aussi non satisfiable. Par conséquent les populations de A et B sont les mêmes. On peut donc fusionner A et B en AB, qui regroupe leurs attributs, rôles et contraintes. Il vient donc :



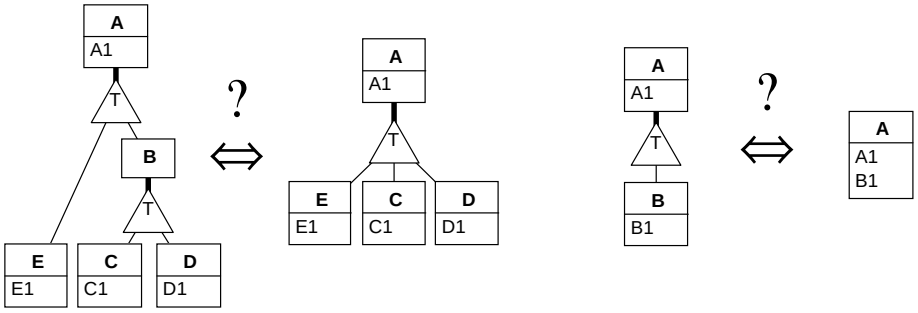
A18.17Il est possible de prouver l'équivalence de deux types d'entités en démontrant qu'ils ont la même population. Montrons-le dans l'exemple ci-dessous (on admet que A et B ont un surtype commun) :



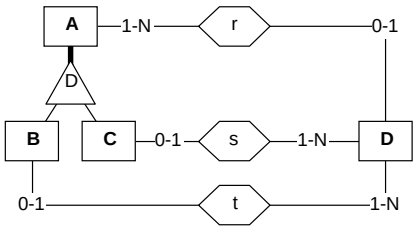
On a : pop(A) = pop(C) ∪ pop(D); pop(B) = pop(C) ∪ pop(D)

d'où : pop(A) = pop(B)

On peut donc admettre que les types d'entités A et B sont identiques. On les renomme AB. Déterminer de la même manière si, dans chacun des deux couples ci-dessous, les schémas sont équivalents.



A18.18Tous les objets du schéma ci-dessous sont-ils satisfiables (d'après [Vigna, 2004]) ?



Solution

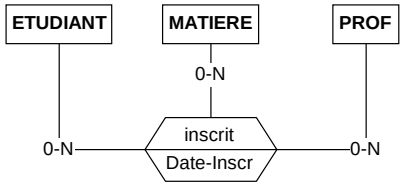
de r on tire : $\mathbf{N}_D \geq \mathbf{N}_A$ de s on tire : $\mathbf{N}_C \geq \mathbf{N}_D$

de t on tire : $\mathbf{N}_B \geq \mathbf{N}_D$ de la relation *is-a* on tire : $\mathbf{N}_A \geq \mathbf{N}_B + \mathbf{N}_C \geq 2 \times \mathbf{N}_D$

on a donc : $\mathbf{N}_A \geq 2 \times \mathbf{N}_A$, ce qui n'est valide que si $\mathbf{N}_A = 0$.

On en conclut qu'aucun des quatre types d'entités n'est satisfiable.

0.19 Normaliser, si nécessaire, le type d'associations ci-dessous[28].



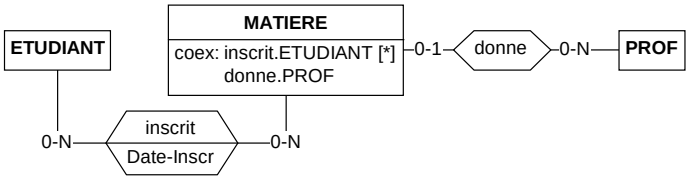
inscrit: MATIERE ⟶ PROFESSEUR

Solution

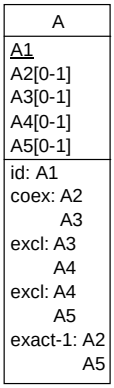
La sémantique interne du type d'associations inscrit est la suivante :

inscrit(ETUDIANT, MATIERE, PROF, Date-Inscr)
inscrit: MATIERE ⟶ PROFESSEUR

La décomposition de ce schéma sous 3e forme normale correspond au schéma ci-dessous. La contrainte de coexistence disparaît si on admet que *toute matière est dispensée par un professeur.*



A18.20 Vérifier la cohérence du schéma ci-dessous et normaliser celui-ci si nécessaire.



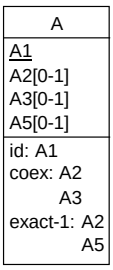
28. Attention, les règles relatives aux identifiants implicites sont de stricte application. En particulier, on tiendra compte de l'identifiant implicite avant de faire intervenir la dépendance fonctionnelle.

Est-il possible qu'une entité A respecte simultanément les 4 contraintes ? Si oui, alors le schéma est cohérent. Il existe plusieurs manières d'aborder cette question. En voici une première qui s'inspire de l'évaluation d'une fonction booléenne complexe par calcul de sa table de vérité.

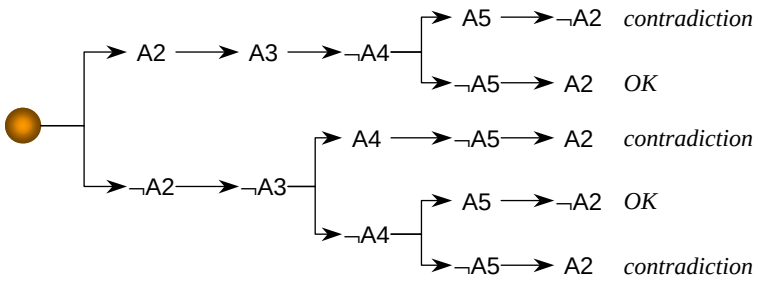
Pour toute entité A, chacun des attributs A2, A3, A4, A5 peut avoir une valeur (ce qu'on note 1) ou non (noté 0). Il existe donc 24 = 16 configurations possibles. Pour chacune d'elles, on évalue chacune des contraintes (VRAI ou FAUX). Les configurations pour lesquelles les 4 contraintes ont la valeur VRAI sont cohérentes. Les autres ne peuvent se produire. Le tableau de calcul est le suivant :

A2	A3	A4	A5	coex:A2,A3	excl:A3,A4	excl:A4,A5	exact1:A2,A5	total
0	0	0	0	VRAI	VRAI	VRAI	FAUX	FAUX
0	0	0	1	VRAI	VRAI	VRAI	VRAI	VRAI
0	0	1	0	VRAI	VRAI	VRAI	FAUX	FAUX
0	0	1	1	VRAI	VRAI	FAUX	VRAI	FAUX
0	1	0	0	FAUX	VRAI	VRAI	FAUX	FAUX
0	1	0	1	FAUX	VRAI	VRAI	VRAI	FAUX
0	1	1	0	FAUX	FAUX	VRAI	FAUX	FAUX
0	1	1	1	FAUX	FAUX	FAUX	VRAI	FAUX
1	0	0	0	FAUX	VRAI	VRAI	VRAI	FAUX
1	0	0	1	FAUX	VRAI	VRAI	FAUX	FAUX
1	0	1	0	FAUX	VRAI	VRAI	VRAI	FAUX
1	0	1	1	FAUX	VRAI	FAUX	FAUX	FAUX
1	1	0	0	VRAI	VRAI	VRAI	VRAI	VRAI
1	1	0	1	VRAI	VRAI	VRAI	FAUX	FAUX
1	1	1	0	VRAI	FAUX	VRAI	VRAI	FAUX
1	1	1	1	VRAI	FAUX	FAUX	FAUX	FAUX

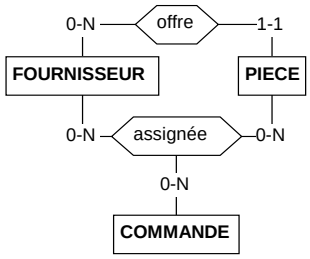
On observe que 2 configurations satisfont simultanément des 4 contraintes. Le schéma est donc cohérent. En revanche, toutes les autres configurations d'attributs sont invalides et n'apparaîtront jamais. On observe en outre que l'attribut A4 n'a jamais de valeur dans les deux configurations valides et qu'il peut donc être supprimé. Le schéma se simplifie comme suit :



On peut proposer une autre méthode d'analyse, plus simple et plus rapide. On construit un arbre définissant, pour la suite des attributs A2, A3, A4, A5, leur état correspondant aux contraintes. On observe les contradictions, selon lesquelles on déduit d'un attribut qui a une valeur le fait qu'il n'en a pas, ou le contraire. Partant de A2, qui soit a une valeur (branche supérieure, notée "A2") soit n'en a pas de valeur (branche inférieure, notée "¬A2"), l'arbre se présente comme ci-dessous. Deux configurations sur les 5 ne présentent pas de contradictions ("OK"). Elles se caractérisent par l'absence de valeur pour A4. On peut alors simplifier le schéma comme ci-dessus.



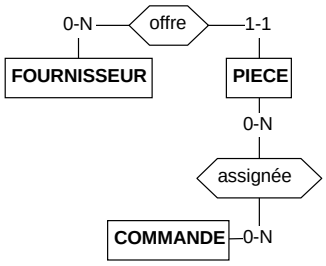
A18.21 Le schéma ci-dessous comporte des constructions contradictoires. Il peut aussi être décrit comme un schéma non normalisé. Proposer une version normalisée et cohérente.



assignée[FOURNISSEUR,PIECE] ⊆ offre[FOURNISSEUR,PIECE]

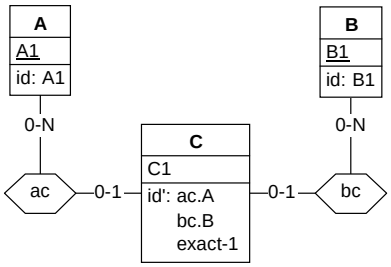
Solution

La contrainte dit qu'il serait stupide d'assigner la commande d'une pièce à un fournisseur qui ne livrerait pas la pièce commandée. Jusque là, le schéma est naturel. On note cependant une particularité intéressante : toute pièce est livrée par un et un seul fournisseur. Comme conséquence de la contrainte d'inclusion, le fournisseur renseigné dans une association assignée n'est rien d'autre que le fournisseur précisé dans le type d'associations offre. Le rôle assignée.FOURNISSEUR est donc inutile. En le supprimant, on obtient le schéma normalisé ci-dessous.



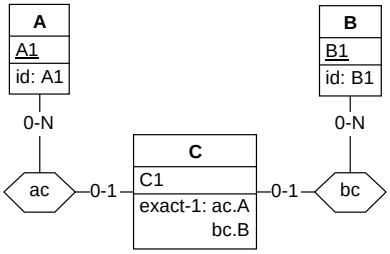
Les concepts étudiés au chapitre 3 peuvent nous aider à raisonner de manière plus rigoureuse. La propriété 6 des dépendance fonctionnelles (*Propagation vers la projection*) ou, de manière équivalente, la propriété 4 des contraintes d'inclusion (*Propagation des DF*), nous permettent de conclure que la DF de offre existe aussi dans assignée. Par conséquent, le type d'associations assignée n'est pas normalisé. On peut le décomposer en deux fragments, dont l'un est redondant avec offre et peut être supprimé.

A18.22Un schéma conceptuel comporte le fragment suivant. Est-il correct ?

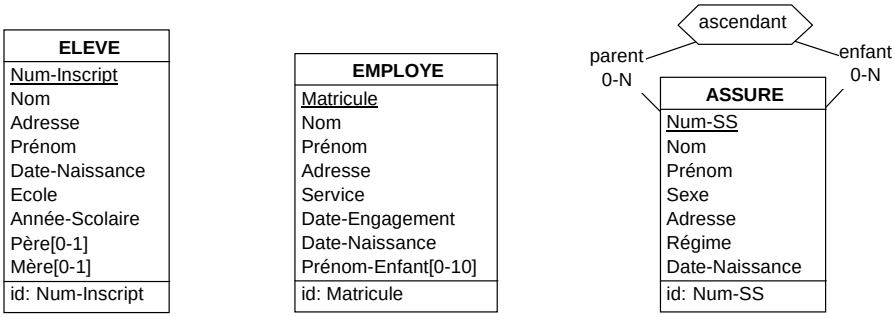


Solution

La contrainte d'exclusion implique qu'aucune entité C n'aura jamais d'identifiant complètement non null, de sorte que l'unicité ne sera jamais évaluée. On peut supprimer l'identifiant de C, inutile :

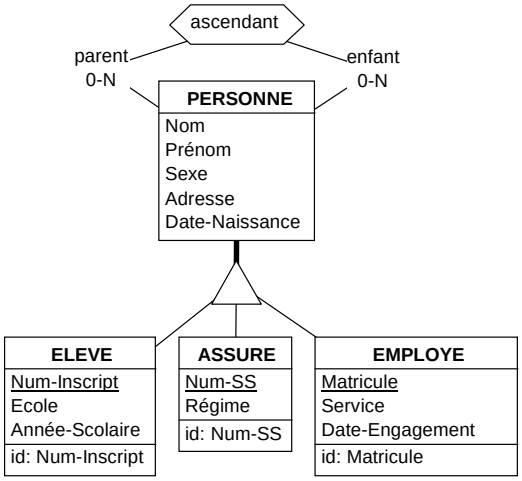


A18.23Proposer un schéma conceptuel minimal intégrant les informations contenues dans les trois types d'entités ci-dessous.



Solution

Un exemple résultant de certains options, qui seraient à discuter :



A18.24Proposer un schéma conceptuel traduisant les concepts de l'énoncé suivant.

Une société scientifique organise depuis plusieurs années une conférence annuelle au mois de septembre. Chaque conférence est gérée par un comité scientifique. La conférence est accueillie par une organisation (université, entreprise ou administration) dont les cordonnées précises sont connues. Le comité scientifique est constitué de membres dont l'un en est nommé le président. Les membres sont décrits par leur nom, leur prénom et leur affiliation. Celle-ci est une organisation. La composition du comité et l'organisation d'accueil changent d'année en année.

Suite à un appel à communications, généralement diffusé en février, la conférence reçoit des articles. Ceux-ci sont numérotés dans le contexte de la conférence. Il n'est pas interdit à des membres du comité scientifique de soumettre des articles. Si un article est écrit par plusieurs auteurs, l'un d'eux

doit se déclarer auteur principal. Chaque article est transmis à plusieurs évaluateurs, qui sont des membres du comité scientifique de la conférence.

Au terme du processus d'évaluation, chaque évaluateur transmet au président du comité, pour chacun des articles examinés, le résultat sous la forme d'une note et d'un commentaire. Après lecture des évaluations de chaque article, le président décide si celui-ci est accepté ou refusé.

Le comité établit ensuite le programme de la conférence. Les articles retenus sont groupés par thèmes. A chaque thème sont associées une ou plusieurs sessions d'une demi-journée. Chaque article fait l'objet d'une présentation qui reçoit une plage horaire lors de sa session et qui est assurée par un de ses auteurs. Un président est assigné à chaque session. Il est choisi parmi les membres du comité scientifique.

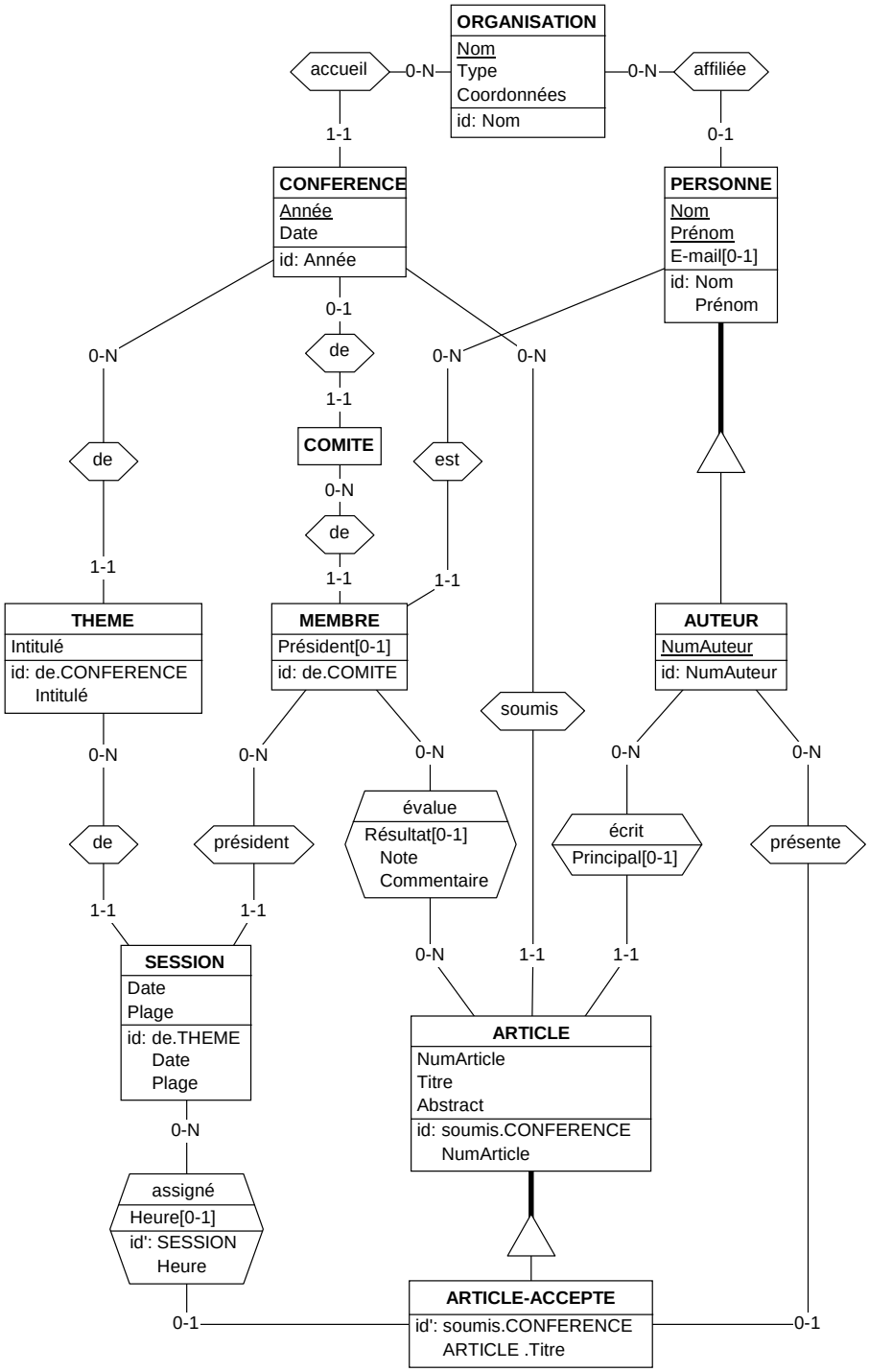
Les auteurs reçoivent un numéro qui les identifie en tant qu'auteurs au sein de la conférence. Un auteur peut signer plusieurs articles. Un article est caractérisé par son titre et son abstract. Les articles acceptés dans une conférence déterminée ont des titres distincts.

Toutes les personnes impliquées dans ces conférences sont caractérisées par un nom, un prénom et éventuellement une adresse électronique. Le nom et le prénom constituent l'identifiant d'une personne.

Solution

Questions à débattre relatives à la solution suggérée :

- quelle est la sémantique externe du type d'entités INTERVENANT ?
- pourrait-on s'en passer ?
- le type d'entités COMITE est-il utile ?
- le type d'entités THEME dépend-il de CONFERENCE ?
- comment indiquer qu'il n'y a qu'un seul président par COMITE ?
- comment indiquer qu'il n'y a qu'un auteur principal par ARTICLE ?
- n'existe-t-il pas de contraintes cycliques dans ce schéma ?


ORGANISATION
Nom
Type
Coordonnées
id: Nom
accueil
0-N
affiliée
1-1
0-1
CONFERENCE
Année
Date
id: Année
PERSONNE
Nom
Prénom
E-mail[0-1]
id: Nom
Prénom
0-1
de
1-1
COMITE
est
soumis
THEME
Intitulé
id: de.CONFERENCE
Intitulé
MEMBRE
Président[0-1]
id: de.COMITE
AUTEUR
NumAuteur
id: NumAuteur
président
évalue
Résultat[0-1]
Note
Commentaire
écrit
Principal[0-1]
présente
SESSION
Date
Plage
id: de.THEME
Date
Plage
assigné
Heure[0-1]
id': SESSION
Heure
ARTICLE
NumArticle
Titre
Abstract
id: soumis.CONFERENCE
NumArticle
ARTICLE-ACCEPTE
id': soumis.CONFERENCE
ARTICLE .Titre