Ce document constitue l'annexe 23 de l'ouvrage "Bases de données", J-L Hainaut, Dunod, 2015

***Date de dernière modification** : 1/12/2011*

Annexe **23**

Études de cas

Cette annexe, rédigée en 2009 pour la 1re édition de l'ouvrage, présente une collection d'études de cas additionnelles illustrant et prolongeant les méthodes et les techniques décrites dans l'ouvrage.

A23.1INTRODUCTION

Le domaine des bases de données est extraordinairement riche et diversifié. L'objectif de cette annexe est d'ouvrir quelques fenêtres sur des applications qui illustrent les principes développés dans l'ouvrage mais aussi de montrer, par des exemples, quelques opportunités d'applications nouvelles.

A ce titre, cette section doit être considérée comme un portefeuille incomplet et évolutif. Chaque étude a pour ambition d'éveiller l'intérêt du lecteur en lui suggérant des pistes de réflexions mais certainement pas de lui fournir des solutions correctes et complètes. On ne s'étonnera pas dès lors que ces études comportent nombres d'erreurs et de maladresses.

Merci d'avance au lecteur qui aura l'amabilité de nous signaler toute erreur et de nous suggérer toute amélioration dans la résolution de ces cas.

A23.2LE PORTE-FEUILLE D'ÉTUDES DE CAS

A cette date (juin 2009), cette annexe comprend neuf études de cas.

Section	Intitulé	Thématique
A23.3	Voyages aériens	Conception d'une base de données
A23.4	Intégration de schémas pour un établissement scolaire	Intégration de schémas
A23.5	Tournées de distribution de colis	Normalisation conceptuelle
A23.6	Représentation des graphes	Modélisation
A23.7	Rétro-ingénierie d'une BD relationnelle	Rétro-ingénierie
A23.8	Gestion de clés	Rétro-ingénierie
A23.9	Migration de fichiers	Rétro-ingénierie; conception
A23.10	Dictionnaire de données	Rétro-ingénierie; normalisation
A23.11	Analyse d'un stack Hypercard	Rétro-ingénierie; treillis de Galois
A23.12	SQL et les ontologies	Modélisation; migration; généralisation

D'autres études viendront s'ajouter dans l'avenir.

A23.3VOYAGES AÉRIENS

Cette étude de cas s'inspire d'un modèle décisionnel utilisé dans un séminaire de formation à destination du personnel d'UTA dans les années quatre-vingt. Son traitement complet sous forme d'une base de données et d'un modèle équationnel est proposé dans [Hainaut, 2005]. On propose ici de développer la base de données.

A23.3.1Énoncé

Le domaine d'application concerne des vols organisés par une compagnie aérienne, et dont on veut déterminer la structure du coût. On considère donc qu'un vol relie deux aéroports en passant par un certain nombre d'escales, qui sont également, du moins quand les choses se passent normalement, des aéroports. Un aéroport porte un nom, mais sera le plus souvent désigné par un code standard propre aux compagnies aériennes. Un même vol peut être effectué à des dates différentes par des appareils différents. Un appareil, désigné par son modèle, est caractérisé par la capacité de ses réservoirs (en kilos de carburant) ainsi que la consommation à vide, équipage compris (en kilos de carburant par km). On connaît aussi sa charge utile maximale et sa consommation supplémentaire par kilo de charge (en kilos de carburant par kilo de charge et par km). Il est à noter cependant que la consommation à vide n'inclut pas le transport du carburant lui-même. On admet que la charge utile (fret et passagers) est constante pour toute la durée du vol, mais qu'elle peut varier d'une

date à l'autre. On connaît la longueur de chaque tronçon du vol, c'est-à-dire la distance entre deux aéroports, ou escales, consécutifs de ce vol. On supposera que la consommation en vol est une fonction linéaire de la charge emportée (charge utile + carburant).

A chaque escale, l'appareil est ravitaillé en carburant. Celui-ci est acheté au tarif local (en dollars par kilo). Le tarif local dépend de la date.

Lorsque l'appareil atterrit à la fin de son vol, ainsi qu'à l'aéroport de départ, ses réservoirs peuvent contenir une quantité résiduelle non consommée lors du parcours du tronçon précédent. Pour effectuer le tronçon suivant, il est généralement nécessaire d'ajouter au réservoir une quantité qui permet d'atteindre l'escale ou l'aéroport suivant. Il est cependant possible d'emporter une quantité supérieure à ce qui est strictement nécessaire. Ce supplément peut être intéressant si le tarif local est particulièrement bas et si le tronçon suivant n'est pas trop long. On fera l'hypothèse que la valeur financière d'une quantité résiduelle est à calculer au tarif de l'endroit où cette quantité est observée, donc à l'atterrissage (= valeur de revente). On notera que l'appareil est présumé avoir consommé la quantité résiduelle à l'aéroport de départ et qu'il faut donc la lui imputer, mais qu'il n'a pas consommé celle qui subsiste après l'atterrissage final, et qu'il ne faut donc pas la lui imputer puisqu'elle n'aura pas servi au vol.

A23.3.2Construction du schéma conceptuel (version de base)

L'objectif de calcul du coût des vols transparaît clairement dans ce texte. Il se traduit par de nombreuses règles d'imputation et de ventilation des composants de ce coût. Nous limitant à la construction de la base de données, il faudra repérer les parties de l'énoncé qui décrivent les données, et donc détecter et écarter les concepts et les règles qui correspondent à des faits et propriétés *dérivables*. Nous les abandonnerons d'autant plus volontiers que leur analyse est précisément l'objectif de la deuxième partie de l'ouvrage précité.

Nous ne détaillerons pas l'analyse de l'énoncé et la construction progressive du schéma conceptuel. Nous nous contenterons de proposer un exemple de solution (A23.3.1), accompagné d'une brève description des types d'entités.

Les types d'entités TYPE-APPAREIL, TYPE-VOL, TYPE-TRONCON et AEROPORT définissent les propriétés communes à tous les vols effectués. Ils représentent des concepts indépendants du temps, du moins à court terme. Les types d'entités VOL, TRONCON et ESCALE-VOL représentent au contraire les faits journaliers concernant les vols effectués et les tarifs locaux du carburant. Ils constituent en quelque sorte la *matérialisation* temporelle respectivement de TYPE-VOL, TYPE-TRONCON et AEROPORT, auxquels ils sont associés.

Une entité **TYPE-APPAREIL** représente un modèle d'appareil. Elle est caractérisée par la dénomination du modèle (Modèle), la capacité maximale des réservoirs (Cap-réservoir), la consommation en vol d'un appareil à vide, équipage compris, mais sans passagers, ni fret, ni carburant, exprimée en kilos de carbu-

rant par km (Consom-vide), la consommation en vol nécessaire au transport d'un kilo de charge sur un km (Consom-charge), la charge maximale qu'un appareil peut emporter (Charge-ut-max). Remarquons qu'on ne représente pas ici l'appareil qui effectue un vol déterminé, mais seulement son modèle. Par exemple, il n'existera qu'une entité APPAREIL dont Modèle = « AIRBUS A330 ».

Une entité **TYPE-VOL** représente un type de vol caractérisé par un code identifiant (ID-Tvol) et par la suite des types de tronçons dont il est constitué (TYPE-TRONCON via composé).

Une entité **AEROPORT** représente un aéroport. Elle est caractérisée par le nom en clair de l'aéroport (Nom) et code de l'aéroport (Code-aéroport).

Une entité **TYPE-TRONCON** représente une section ininterrompue d'un type de vol. Elle est attachée à une entité TYPE-VOL, et porte un numéro d'ordre parmi les types de tronçons du type de vol (Num-ordre), la longueur du tronçon en km (Distance), l'aéroport d'origine (AEROPORT via origine-tt) et l'aéroport de destination (AEROPORT via destin-tt).

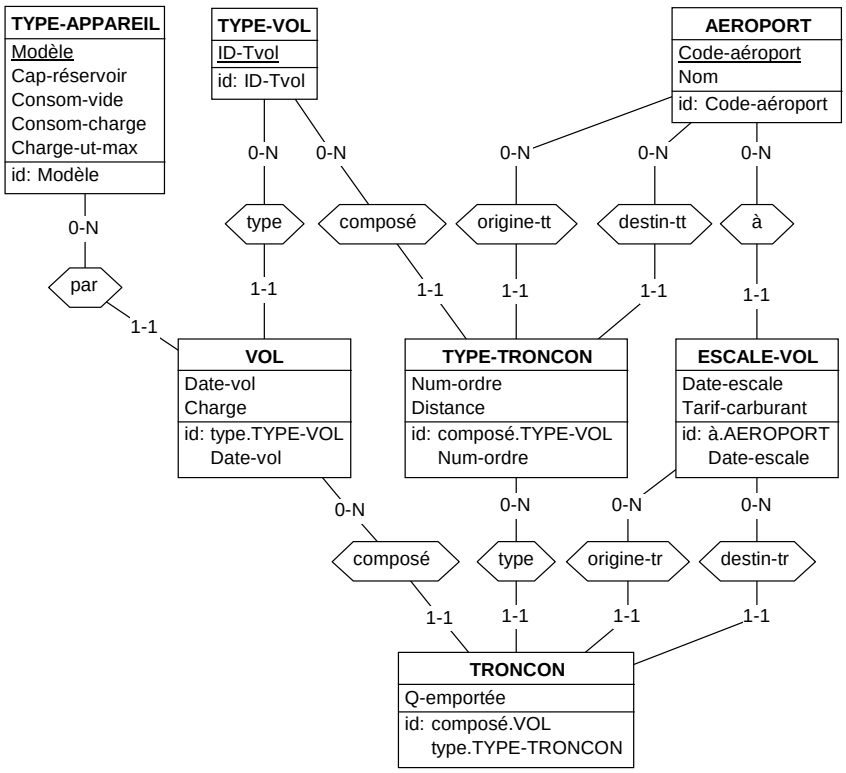


Figure A23.3.1 - Schéma conceptuel de la base de données des vols

Une entité **VOL** représente un vol réel effectué à une date déterminée. Elle correspond à une entité TYPE-VOL qui en définit les propriétés stables. Elle est caractérisée par le modèle de l'appareil effectuant le vol (TYPE-APPAREIL *via* par), la date du vol (Date-vol), la charge utile transportée (Charge), les tronçons dont le vol est constitué (TRONCON via composé).

Une entité **ESCALE-VOL** représente une escale effectuée à un aéroport à une date déterminée. Elle est caractérisée par l'aéroport (AEROPORT), la date à laquelle l'aéroport est considéré (Date-escale), le tarif du carburant dans cet aéroport à cette date, en $US, par kilo (Tarif-carburant).

Une entité **TRONCON** représente une section effectuée par un vol réel. Elle correspond à une entité TYPE-TRONCON qui en définit les propriétés stables. Elle est caractérisée par le vol réel durant lequel le tronçon a été effectué (VOL via composé), la quantité de carburant emportée (achetée) à l'escale de départ du tronçon (Q-emportée), l'escale de départ du tronçon (ESCALE-VOL via origine-tr), l'escale d'arrivée du tronçon (ESCALE-VOL via destin-tr).

A23.3.3Construction du schéma conceptuel (version étendue)

Le schéma conceptuel de la A23.3.1 semble d'une grande simplicité : sept types d'entités et dix types d'associations 1:N. Il est en réalité plus complexe qu'il n'y paraît.
En particulier :

1. on y reconnaît une variante du pattern typique de types d'associations de *composition* et de *matérialisation* de la figure figure F.15.50,
2. ce schéma comporte un nombre important de circuits, dont certains sont certainement le siège de contraintes cycliques (section figure F.15.12.3); ainsi, le TYPE-VOL du VOL d'un TRONCON doit-il être le TYPE-VOL du TYPE-TRONCON de ce même TRONCON,
3. les deux ESCALE-VOLs d'un TRONCON sont certainement différentes, il en est de même des AEROPORTs d'un TYPE-TRONCON,
4. comment peut-on déterminer des AEROPORTs d'origine et de destination d'un TYPE-VOL ? et d'un VOL ? Sont-ils différents ?
5. les attributs Date-vol et Date-escale devraient certainement faire l'objet de contraintes d'intégrité
6. il en est sans doute de même des attributs Cap-réservoir et Q-emportée.

L'identification et l'expression de ces contraintes sont laissées à l'initiative du lecteur.

A23.3.4Production du schéma de tables

L'application des règles de traduction du chapitre figure F.13 conduit au schéma de la A23.3.2. La définition des structures physiques et la traduction en code SQL ne posent pas de problèmes particulier selon la procédure simplifiée du chapitre figure F.13.

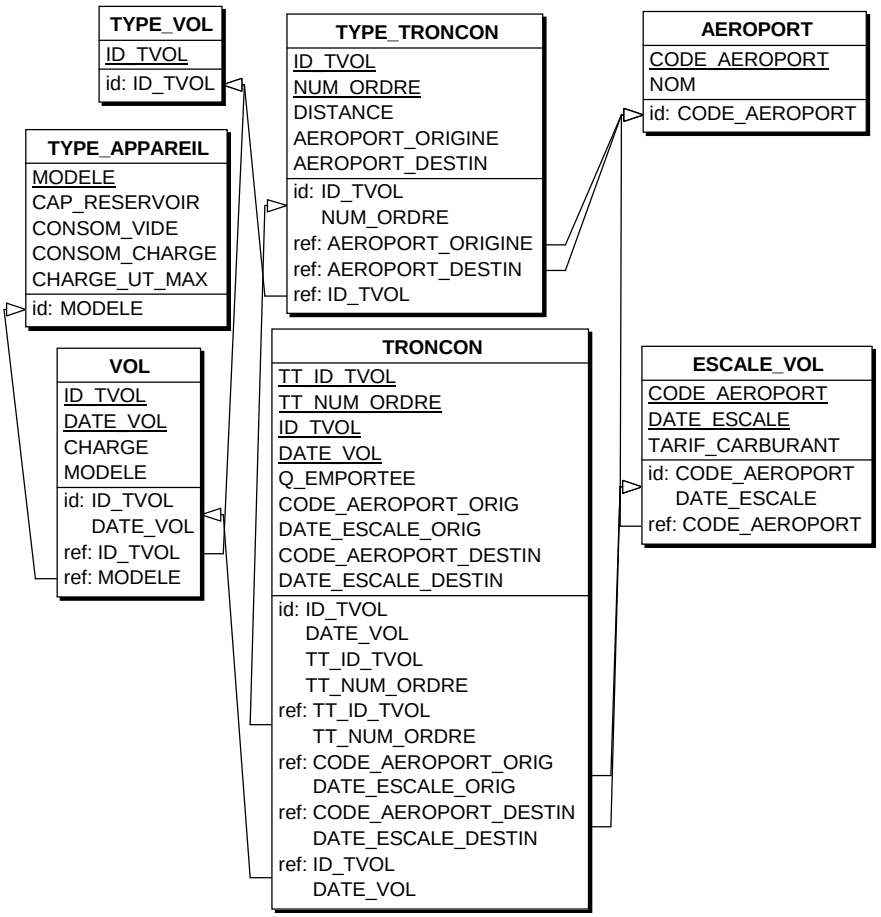


Figure A23.3.2 - Le schéma des tables correspondant au schéma conceptuel F.A23.3.1

A23.4INTÉGRATION DE SCHÉMAS POUR UN ÉTABLISSEMENT SCOLAIRE

Une analyse des besoins d'une fédération d'établissements d'enseignement a conduit aux quatre sous-schémas de la A23.4.1, dont on peut supposer qu'ils décrivent des documents issus de différents sous-systèmes. Proposer un schéma unique qui

reprenne, sans redondance, les concepts sous-jacents à ces quatre sous-schémas.

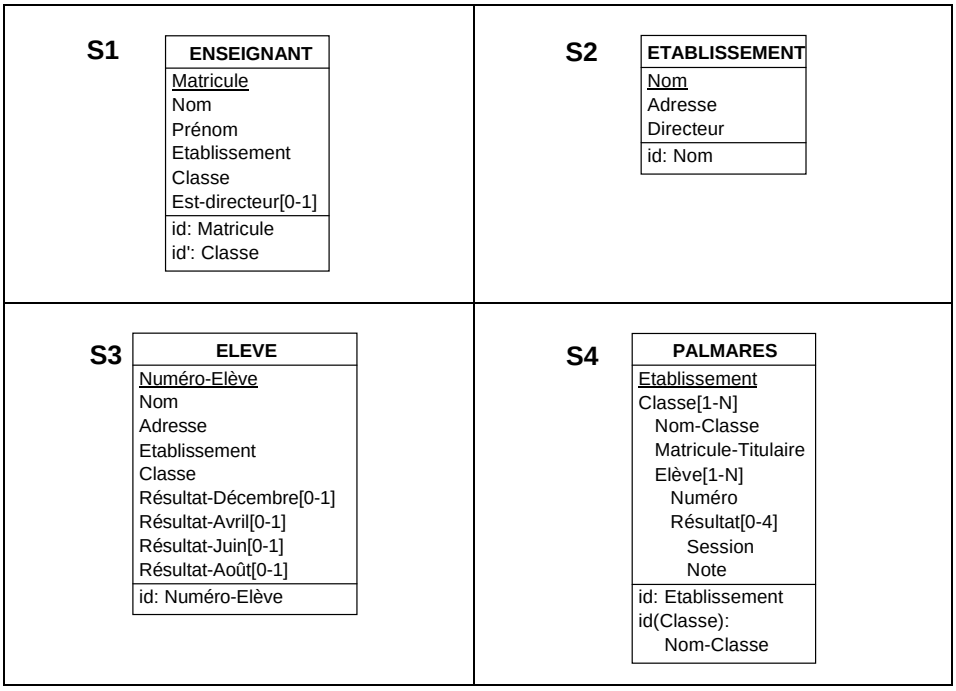


Figure A23.4.1 - Intégration de schémas pour un établissement scolaire : les quatre schémas de base.

Observons d'abord que l'énoncé va un peu vite en besogne. Il apparaît clairement que certains de ces schémas ne sont pas normalisés. Procéder à leur intégration dans leur état actuel nous compliquerait inutilement la tâche. Nous allons donc d'abord les normaliser.

A23.4.1Normalisation des sous-schémas

Pour chaque schéma, nous identifierons les défauts de normalisation puis nous les corrigerons.

a) Normalisation du schéma S1

A priori, le type d'entités ENSEIGNANT ne semble pas nécessiter de transformation de normalisation. Cependant, une analyse plus fine de l'attribut Est-directeur montre que celui-ci joue le rôle d'un *indicateur de sous-type*. Il constituerait donc une *construction insuffisamment expressive*. On suggère de rendre explicite le concept de directeur (A23.4.2).

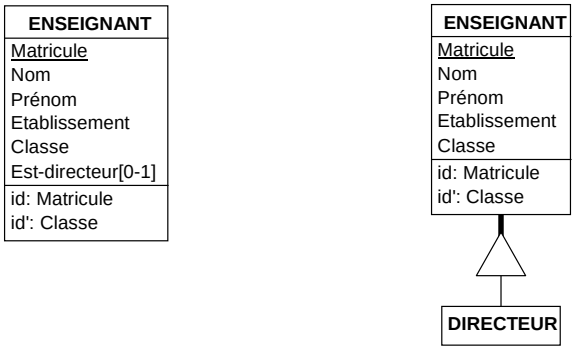


Figure A23.4.2 - Normalisation du schéma S1

b) Normalisation du schéma S2

Ce schéma ne contient pas de défaut particulier de normalisation. Il est conservé tel quel.

c) Normalisation du schéma S3

Le schéma S3 comporte des *attributs sériels*, c'est-à-dire une suite d'attributs présentant une similitude de nom, de type et/ou de longueur suggérant une certaine corrélation sémantique. Les attributs sériels du schéma S3 sont repérables par leurs noms : Résultat-Décembre, Résultat-Avril, Résultat-Juin, Résultat-Août. Ils représentent une collection de résultats identifiés par le mois d'une session d'évaluation. Cette construction est typique de la conception logique relationnelle, résultant de la transformation dite d'*instanciation*. Une construction propre à un niveau d'abstraction qui apparaît à un niveau d'abstraction supérieur est dite *construction étrangère*. On suggère de transformer cette série d'attributs en un attribut complexe (A23.4.3, schéma central). Ce dernier est ensuite transformé en type d'entités (A23.4.3, schéma de droite), auquel il faudrait ajouter la définition du domaine de Session.

Les autres composants de ELEVE ne posent pas de problèmes.

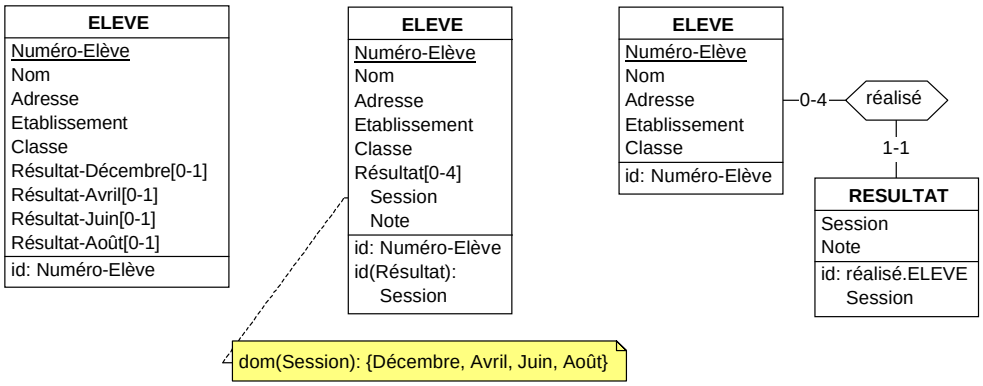


Figure A23.4.3 - Normalisation du schéma S3

d) Normalisation du schéma S4

L'attribut complexe Classe représente de manière évidente un concept majeur qui sera plus à l'aise dans le statut de type d'entités (construction *insuffisamment expressive*). On transforme cet attribut en type d'entités (A23.4.4).

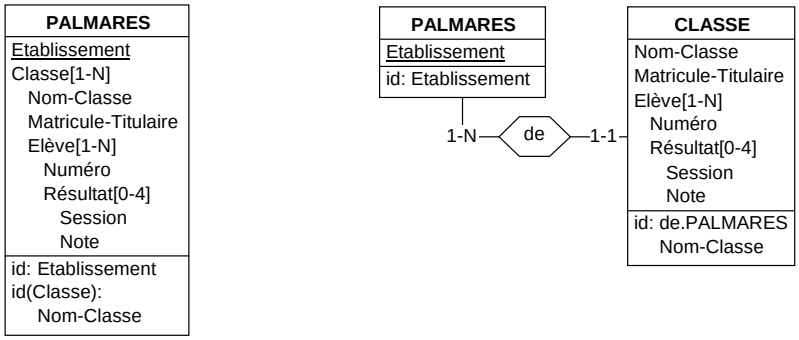


Figure A23.4.4 - Explicitation du type d'entités CLASSE

Le type d'entités CLASSE comporte encore un attribut complexe, Elève, qu'on transforme de la même manière en un type d'entités ELEVE. Remarquons cependant que cet attribut présente une anomalie : n'ayant pas d'identifiant explicite, l'identifiant par défaut est {Numéro, Résultat}, ce qui est non seulement interdit dans le modèle Entité-association (un attribut multivalué ne peut apparaître que dans un identifiant mono-composant), mais en outre peu plausible : on ne voit pas en quoi ses résultats contribuent à l'identification d'un élève ! Par prudence, nous ne définissons pas d'identifiant pour l'instant. Le type d'entités ELEVE possède un attribut complexe, Résultat, que nous transformons également en type d'entités RESULTAT. On propose donc le schéma A23.4.5. Ici non plus, l'identifiant de RESULTAT n'est pas défini.

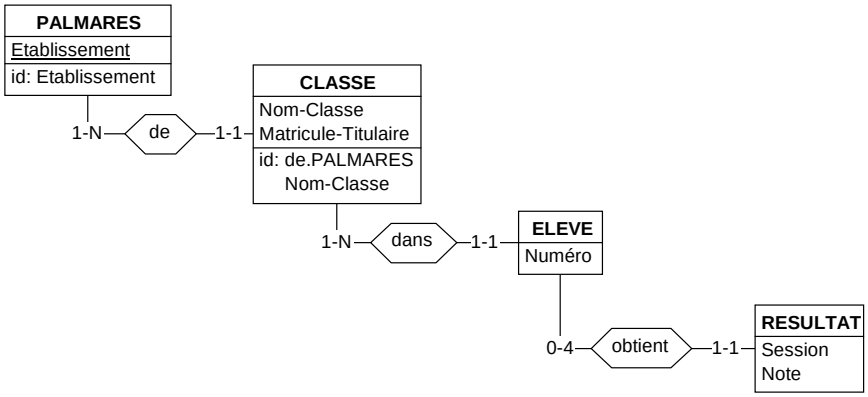


Figure A23.4.5 - Normalisation du schéma S4

A23.4.2Intégration des sous-schémas S1 et S2

Comparons les schémas normalisés S1 (gauche) et S2 (droite) dans la A23.4.6. On observe que le concept représenté par le type d'entités DIRECTEUR est exactement (*égalité*) celui que représente l'attribut Directeur. A l'inverse, le type d'entités ETABLISSEMENT *comprend* l'attribut Etablissement. On établit donc les deux correspondances de la A23.4.6.

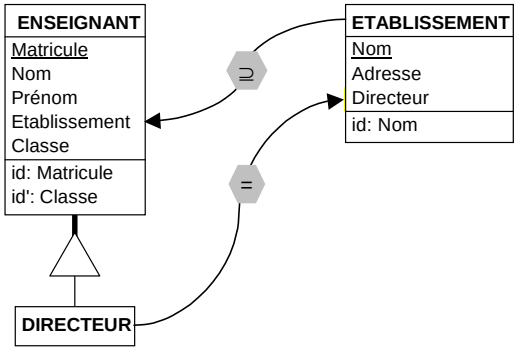


Figure A23.4.6 - Identification des correspondances entre S1 et S2

Ces deux correspondances étant hétérogènes, on transforme les schémas de manière à rendre ces correspondances homogènes (A23.4.7).

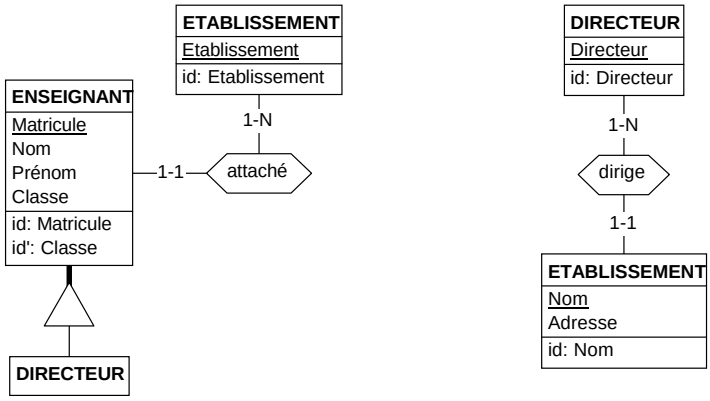


Figure A23.4.7 - Transformation d'homogénéisation des correspondances entre S1 et S2

La fusion des schémas S1 et S2 ainsi homogénéisés est immédiate. Elle produit le schéma S12 (A23.4.8).

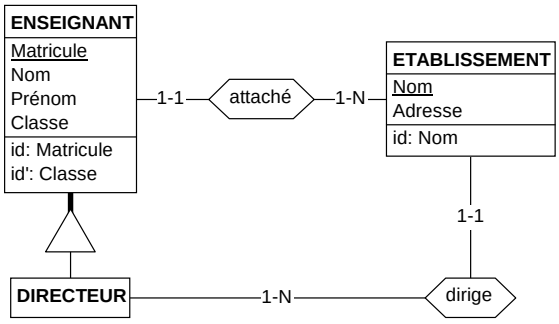


Figure A23.4.8 - Fusion des schémas S1 et S2 (= S12)

A23.4.3Intégration des sous-schémas S12 et S3

On établit une correspondance de *compréhension* entre le type d'entités ETABLISSEMENT (A23.4.8) et l'attribut Etablissement (A23.4.3). Cet attribut est transformé en type d'entités ETABLISSEMENT.

En outre, les deux attributs Classe sont en correspondance d'*égalité*. Il ne s'agit cependant pas d'une correspondance homogène puisque les attributs appartiennent à des types d'entités différents. On transforme chacun de ces attributs en un type d'entités nommé CLASSE.

La fusion des deux schémas ainsi traités est immédiate. Le résultat, nommé S123, est proposé à la A23.4.9.

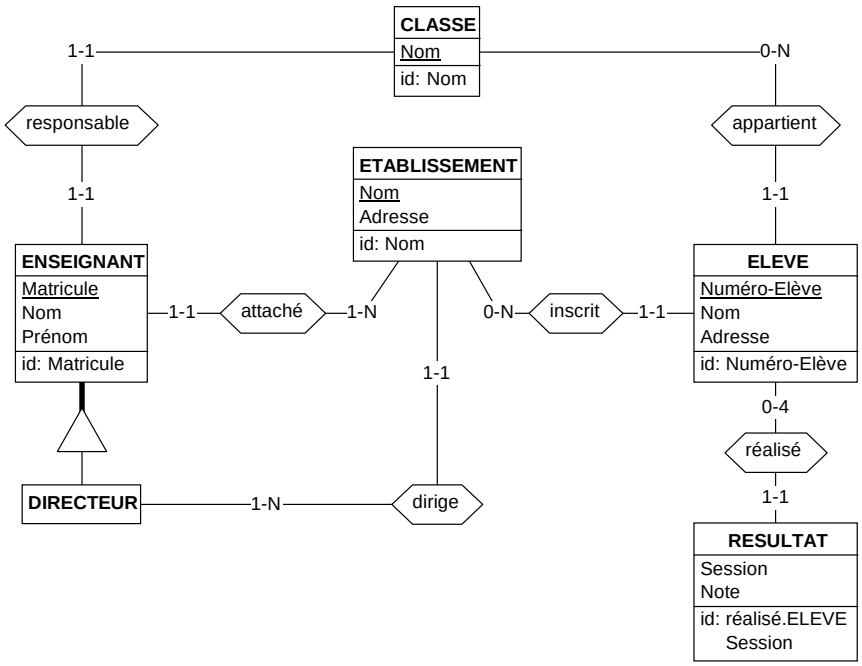


Figure A23.4.9 - Schéma S123 résultant de l'intégration des schémas S1, S2 et S3

A23.4.4Intégration des sous-schémas S123 et S4

Il reste à intégrer les schémas S123 (A23.4.9) et S4 (A23.4.5). On établit les correspondances suivantes :

1. l'attribut Etablissement, identifiant de PALMARES (S4), est en correspondance d'*égalité* avec le type d'entités ETABLISSEMENT (S123); on homogénéise en exprimant cet attribut sous la forme du type d'entités ETABLISSEMENT; ce dernier étant en bijection avec PALMARES, on fusionne ces deux type d'entités au profit d'ETABLISSEMENT;
2. les deux types d'entités CLASSE sont en correspondance d'*égalité*;
3. l'attribut Matricule-titulaire et le type d'associations responsable sont en correspondance d'égalité;
4. un identifiant de ELEVE est suggéré dans le schéma S123; on l'adopte pour le schéma final;
5. un identifiant de RESULTAT est suggéré dans le schéma S123; on l'adopte pour le schéma final;

On obtient ainsi le schéma S1234 brut de la A23.4.10.

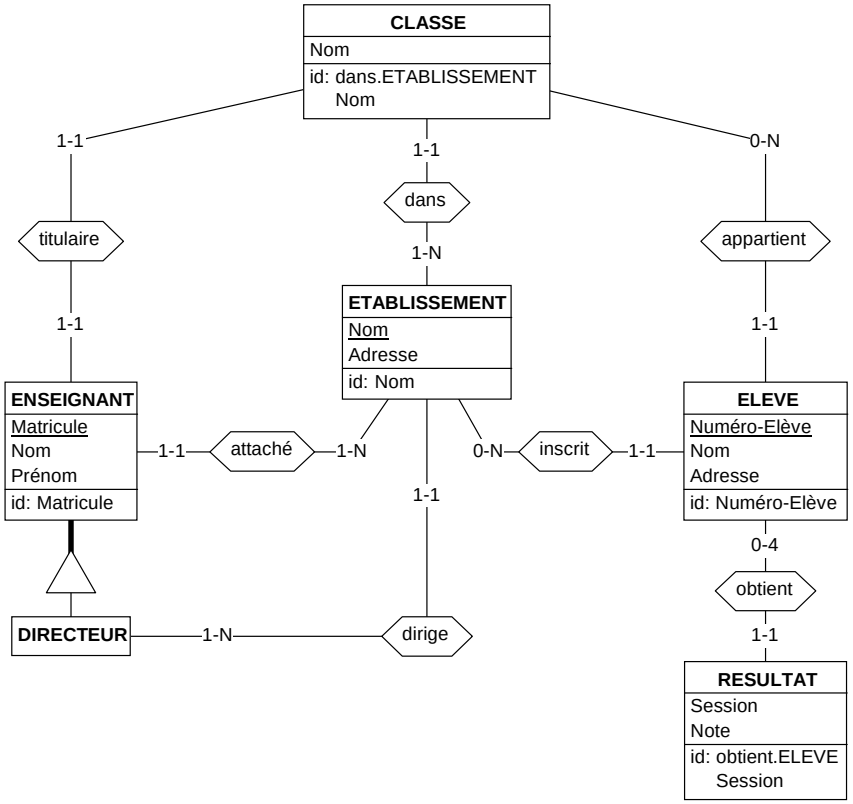


Figure A23.4.10 - Schéma brut S1234 résultant de l'intégration des schémas S123 et S4

A23.4.5Normalisation du schéma intégré

Le schéma intégré comporte généralement des défauts de normalisation. On identifie dans le schéma S1234 des constructions qu'il est possible d'améliorer.

1. Le type d'associations inscrit entre ELEVE et ETABLISSEMENT est redondant il est en effet la composition de appartient et dans. On le supprime.
2. Les cardinalités 1-N de dans.ETABLISSEMENT et attaché.ETABLISSEMENT sont probablement des *sur-spécifications* (*contraintes trop fortes*). On les transforme en 0-N.
3. Les cardinalités 1-1 de titulaire sont également des *contraintes trop fortes*. On les réduit à 0-1.
4. La cardinalité de dirige.ETABLISSEMENT 1-1 est une contrainte trop forte. On la transforme en 0-1.

5. Remarquons d'ailleurs la présence de trois circuits impliquant les types d'entités CLASSE, ETABLISSEMENT et ENSEIGNANT dont tous les rôles sont obligatoires. Il ne s'agit pas à proprement parler de constructions non satisfiables (il est possible d'envisager des populations non vides) mais plutôt des constructions dont la gestion sera très complexe. Les propositions de relaxation ci-dessus sont donc pleinement justifiées.
6. Le sous-type DIRECTEUR apporte peu d'information : sa seule spécificité est de jouer le rôle dirige.DIRECTEUR. On le supprime sans perte d'information. ENSEIGNANT reprend ce rôle, qui devient facultatif.
7. Dans le type d'entités RESULTAT, l'attribut Session est perçu comme important. On le transforme en type d'entité SESSION.
8. Suite à cette transformation, le type d'entités RESULTAT apparaît comme un *type d'entités association*. On peut envisager de la transformer en un type d'associations résultat.
9. On procède à quelques modifications de noms.
10. On reprend le domaine de valeurs de Date (non illustré).

On obtient ainsi le schéma intégré normalisé de la A23.4.11.

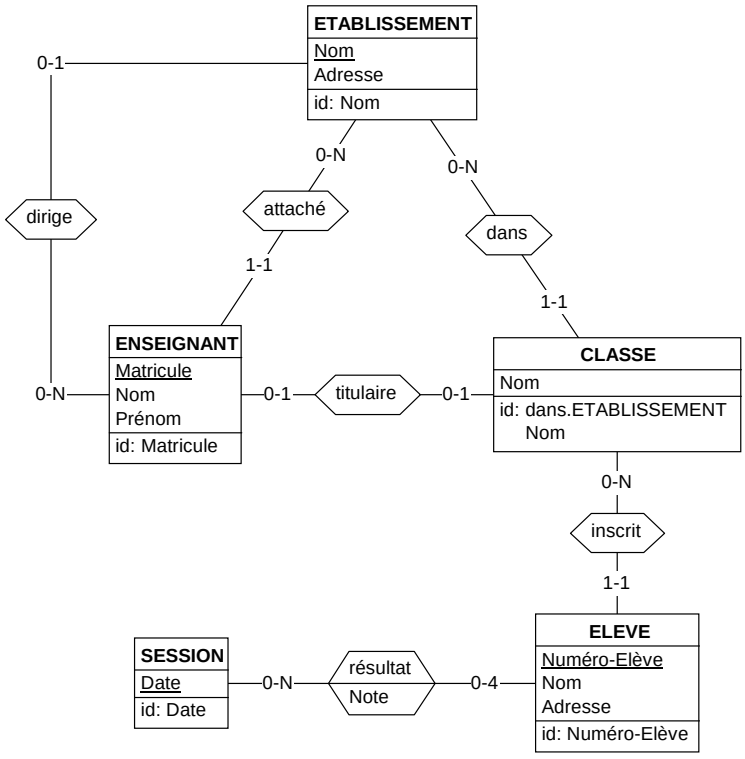


Figure A23.4.11 - Normalisation du schéma S1234

A23.5 TOURNÉES DE DISTRIBUTION DE COLIS

Construire un schéma conceptuel représentant les concepts sous-jacents à l'énoncé suivant.

Une entreprise de distribution dispose de véhicules effectuant des tournées journalières. Un véhicule de sortie effectue une et une seule tournée par jour ouvrable. Une tournée est effectuée entièrement le même jour, par un seul véhicule. Durant une tournée, le véhicule qui l'effectue distribue des colis. A la fin d'une tournée, tous ses colis ont été distribués.

En apparence anodin, cet énoncé pose quelques problèmes intéressants, dont la résolution exploite les résultats de la théorie de la normalisation étudiée au chapitre figure F.3.

a) Première analyse

Une lecture en diagonale des propositions de l'énoncé peut conduire au schéma de la A23.5.1 :

- le type d'association effectue traduit le fait qu'*un véhicule effectue une tournée par jour ouvrable*,
- le type d'association distribue exprime le fait que *durant une tournée, le véhicule qui l'effectue distribue des colis*.

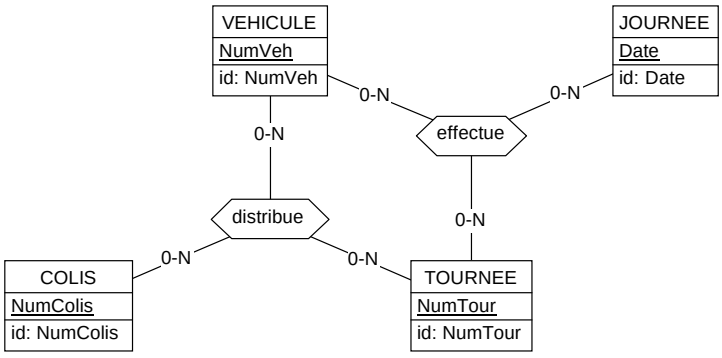


Figure A23.5.1 - Les types d'entités et les types d'associations.

b) Etude des cardinalités

Les cardinalités [0-N] ont été assignées à titre indicatif. Analysons-les d'une manière plus précise.

- Type d'associations effectue :
  - effectue.VEHICULE : un véhicule peut avoir effectué un nombre quelconque de tournées, ce qui confirme la cardinalité [0-N].

  - effectue.JOURNEE : pendant une journée ouvrable déterminée, un nombre quelconque (mais supérieur à 0) de tournées peuvent être effectuées; on suggère la cardinalité [1-N].
  - effectue.TOURNEE : une tournée est effectuée par un seul véhicule durant une seule journée; d'où la cardinalité [1-1].
- Type d'associations distribue :
  - distribue.VEHICULE : un véhicule distribue un nombre quelconque de colis pendant un nombre quelconque de tournées; cardinalité [0-N].
  - distribue.TOURNEE : pendant une tournée le véhicule distribue au moins un colis; cardinalité [1-N].
  - distribue.COLIS : un colis est (ou sera) distribué durant une tournée effectuée par un véhicule; cardinalité [0-1].

Le schéma est corrigé sous la forme de la A23.5.2.

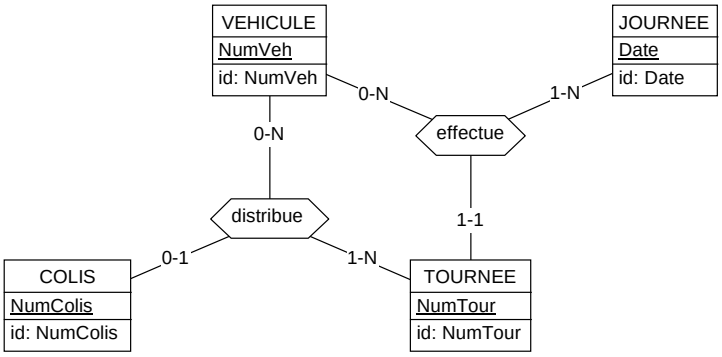


Figure A23.5.2 - Définition des cardinalités correctes.

c) Contraintes supplémentaires

L'énoncé nous permet de mettre en lumière deux propriétés supplémentaires :

1. un véhicule de sortie effectue une et une seule tournée par jour

   étant donné un véhicule et une journée, il n'y correspond pas plus d'une tournée; on a donc :

   effectue: VEHICULE,JOURNEE $\longrightarrow$ TOURNEE

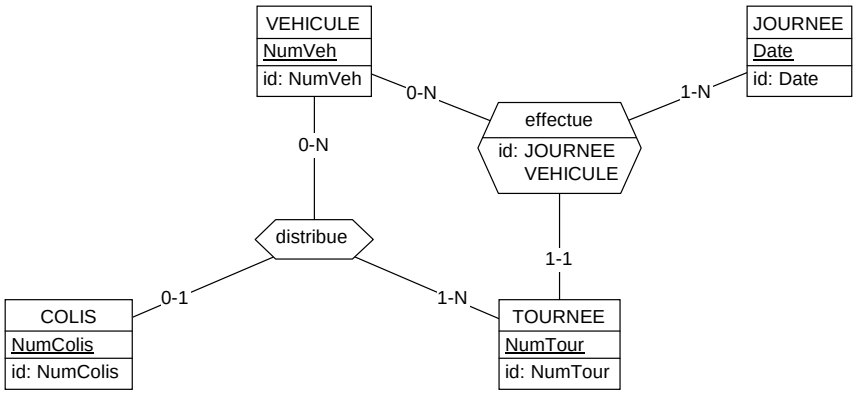
   En d'autres termes, {VEHICULE,JOURNEE} est un identifiant de effectue.

2. durant une tournée, le véhicule qui l'effectue distribue des colis

   le véhicule qui distribue un colis pendant une tournée est celui qui effectue cette tournée; on peut donc écrire également :

   distribue[VEHICULE,TOURNEE] $\subseteq$ effectue[VEHICULE,TOURNEE]

On complète le schéma comme suit.



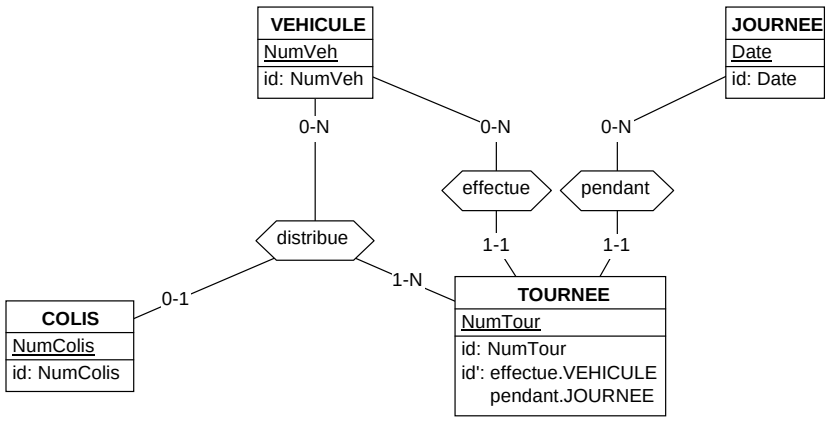
distribue[VEHICULE,TOURNEE] $\subseteq$ effectue[VEHICULE,TOURNEE]

Figure A23.5.3 - Contraintes sur les types d'associations.

d) Normalisation du schéma

Le schéma ainsi obtenu peut être considéré comme correct et complet. Il doit cependant être normalisé.

1. La cardinalité [1-1] du rôle effectue.TOURNEE nous autorise à décomposer effectue en deux types d'associations binaires 1:N. L'identifiant explicite de effectue est transféré vers TOURNEE. La contrainte d'inclusion est ajustée.
   Le schéma est transformé selon la A23.5.4.



distribue[VEHICULE,TOURNEE] $\subseteq$ effectue[VEHICULE,TOURNEE]

Figure A23.5.4 - Décomposition du type d'associations *effectue*.

La contrainte d'inclusion définie sur distribue et effectue induit une propriété intéressante : la dépendance fonctionnelle effectue:TOURNEE $\longrightarrow$ VEHICULE

est également présente dans tout sous-ensemble de effectue, et donc en particulier dans distribue; on a donc :

distribue: TOURNEE ⟶ VEHICULE

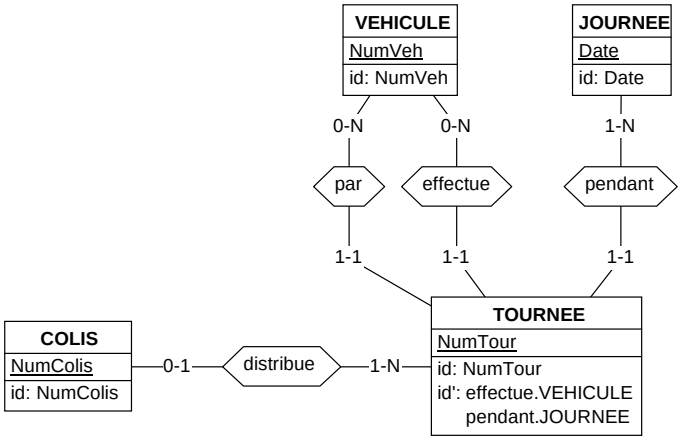
Dans le type d'associations distribue(VEHICULE,TOURNEE,COLIS), TOURNEE apparaît ainsi comme un déterminant qui n'est pas un identifiant. distribue n'est donc pas normalisé et doit être décomposé comme suit :

distribue(TOURNEE,COLIS)
par(TOURNEE,VEHICULE)
par[TOURNEE] = distribue[TOURNEE]

Cette dernière contrainte est garantie par la cardinalité minimale 1 de par et de distribue. La contrainte d'inclusion entre distribue et effectue est ajustée comme suit :

par ⊆ effectue

La A23.5.5 représente le schéma résultant.



par[VEHICULE,TOURNEE] ⊆ effectue[VEHICULE,TOURNEE]

Figure A23.5.5 - Décomposition du type d'associations *distribue* suite à la propagation de la DF de *effectue*.

2. On interprète la contrainte par ⊆ effectue et les cardinalités [1-1] de effectue.TOURNEE et par.TOURNEE comme suit : *toute entité TOURNEE apparaît dans une association effectue et dans une association par* ***qui sont identiques***. La contrainte d'inclusion est donc une contrainte d'égalité, qui nous autorise à supprimer l'un de ces types d'associations, soit par.
3. En outre, le type d'entités JOURNEE représente manifestement un concept mineur, qui pourrait plus simplement être représenté par un simple attribut de TOURNEE sans perte de sémantique.

On obtient ainsi le schéma final normalisé de la A23.5.6, désespérément simple !

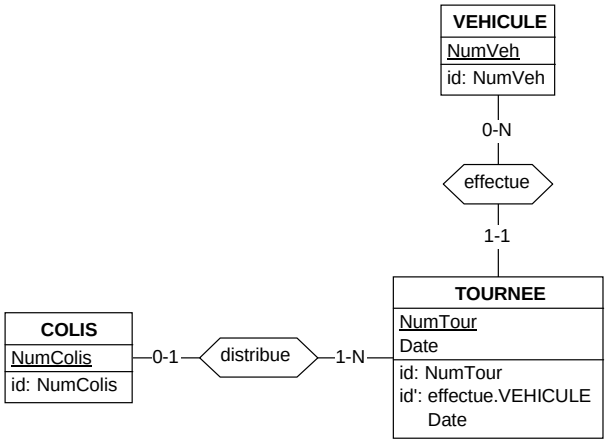


Figure A23.5.6 - Tout ça pour ça ....

A23.6REPRÉSENTATION DES GRAPHES

De nombreux schémas incluent la représentation de structures qui sont fondamentalement des graphes, c'est-à-dire des ensembles de noeuds munis chacun d'une relation binaire. Ainsi en est-il des nomenclatures de produits, des relations hiérarchiques dans une organisation, des flux de documents entre agents, des divers réseaux de distribution d'énergie, de téléphonie ou de distribution de fluides (eau, gaz, pétrole, etc.), ou encore des réseaux routiers et ferroviaires, des lignes aériennes, des plans et des cartes, des circuits électroniques.

Il n'est donc pas inutile d'étudier la modélisation des divers types de graphes indépendamment des aspects spécifiques du domaine d'application qu'ils représentent.

A23.6.1Rappel des principales notions

- **Graphe**

  Un graphe est une structure constituée d'un ensemble N d'éléments appelés **noeuds** ou sommets et d'un ensemble A d'**arcs**, ou arêtes, reliant chacun deux noeuds (non nécessairement distincts). L'ensemble A représente une relation binaire entre N et lui-même. Formellement, un graphe G est défini comme suit :

  $G \equiv (N,A)$, *où* $A \subseteq NxN$

- **Graphe orienté et non orienté**

  Un *graphe est orienté* si chaque arc est défini par un *couple* (n1,n2) dont chaque élément joue un rôle spécifique. On dira que l'arc part d'un noeud dit *origine* et aboutit à un noeud dit *cible*.

  Dans un *graphe non orienté*, un arc[1] est défini par un *ensemble* {n1,n2} de deux noeuds qui ne jouent pas de rôle particulier l'un par rapport à l'autre.

  Dans un graphe orienté, il ne peut exister plus d'un arc d'un même noeud vers un autre noeud Dans un graphe non orienté, il ne peut exister plus d'un arc entre deux noeuds déterminés.

  Un graphe non orienté peut aussi être perçu comme un graphe orienté *symétrique*, c'est-à-dire dans lequel à tout arc (n1,n2) correspond l'arc inverse (n2,n1). Cette manière de voir complique cependant les concepts de cycle et d'arbre.

  L'orientation d'un arc a pour but de distinguer le rôle et les caractéristiques de chacune de ses extrémités. Elle ne définit pas nécessairement un sens de parcours imposé. Rien n'empêche de parcourir les arcs d'un graphe *dans le sens inverse* de leur orientation.

Figure A23.6.1 - Graphe orienté (gauche) et non orienté (droite)

- **Degré d'un noeud**

  Le degré d'un noeud n est nombre d'arcs dont n est une extrémité. Dans un graphe orienté, on parle aussi de *degré entrant* (nombre d'arcs dont n est cible) et de *degré sortant* (nombre d'arcs dont n est origine). Un noeud qui n'apparaît dans aucun arc est dit *isolé*. Il est de degré 0. Un noeud non isolé dont le degré entrant est 0 est un *noeud source*. Un noeud non isolé dont le degré sortant est 0 est un *noeud puits*.

1. Un arc non orienté est parfois appelé arête.

- **Chaîne, chemin, longueur**

  Dans un graphe non orienté ou orienté, une *chaîne* est toute suite d'au moins deux noeuds (n1, ..., nt) telle qu'il existe un arc (dont l'orientation est indifférente) entre chaque paire de noeuds successifs de la chaîne.
  Dans un graphe orienté, un *chemin* est toute suite de t>1 noeuds (n1, ..., nt) telle qu'il existe, entre chaque paire de noeuds (ni,nj) consécutifs de la suite, un arc d'origine ni et de cible nj.
  La *longueur* d'une chaîne (d'un chemin) est le nombre d'arcs ainsi définis, ou encore t-1.

- **Chaîne (chemin) simple**

  Une chaîne (chemin) est *simple* si un arc n'y apparaît pas plus d'une fois. Une chaîne peut cependant passer plus d'une fois par un noeud déterminé.

- **Chaîne (chemin) élémentaire**

  Une chaîne (chemin) est *élémentaire* si les noeuds de sa suite sont distincts. Autrement dit, une chaîne (un chemin) élémentaire ne passe pas plus d'une fois par un noeud déterminé. Une chaîne élémentaire est forcément simple.

- **Connexité**

  Un graphe non orienté est *connexe* si, pour toute paire de noeuds distincts, il existe une chaîne à laquelle ces noeuds appartiennent. La définition est identique pour un graphe orienté, pour autant qu'on fasse abstraction du sens des arcs.
  Une *composante connexe* C d'un graphe G est un graphe connexe constitué d'un sous-ensemble de noeuds de G et de tous les arcs dont les deux extrémités sont dans C, et tel que tout ajout d'un autre noeud de G lui ferait perdre la propriété de connexité. Un graphe connexe est formé d'une seule composante connexe.

- **Cycle et circuit**

  Un *cycle* est une chaîne simple de mêmes extrémités : n1= nt. Un *circuit* est un chemin de mêmes extrémités : en parcourant les noeuds d'un circuit, on aboutit à son noeud de départ.

- **Graphe sans cycle ou acyclique**

  Un graphe non orienté est dit *sans cycle* ou *acyclique* s'il ne contient aucun cycle. En particulier, il n'existe pas plus d'une chaîne entre deux noeuds distincts quelconques.

  .

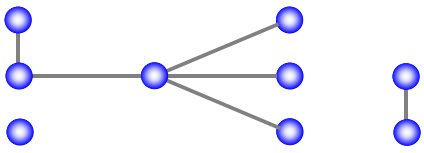


Figure A23.6.2 - Graphe acyclique à trois composante connexe

Un *graphe orienté* est dit *acyclique* si, considérant les arcs comme non orientés, il est acyclique

- **Graphe sans circuits**
  Un graphe orienté est dit *sans circuits* s'il ne comporte aucun circuit. Notons qu'il n'est pas nécessairement acyclique. Un graphe sans circuit est aussi appelé *hiérarchique*. Remarquons que, par abus de langage, les graphes orientés sans circuits sont souvent appelés *graphes orientés acycliques*[2].

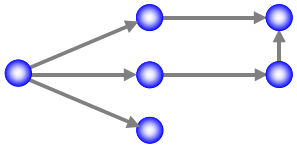


Figure A23.6.3 - Graphe avec cycle mais sans circuit

- **Arbre non orienté**
  Un graphe non orienté est un *arbre* s'il est connexe et acyclique. Tout noeud peut être choisi comme la *racine* de l'arbre[3]. Entre deux noeuds quelconques d'un arbre, il existe une et une seule chaîne élémentaire.
  Un *graphe acyclique* est constitué d'un *ensemble d'arbres*, dit aussi *forêt*.
  Le *niveau d'un noeud* quelconque d'un arbre est égal à 1 + la longueur de la chaîne qui le relie à la racine. Le *niveau* de la racine est de 1.

- **Arbre orienté**
  Un graphe orienté est un *arbre* s'il est connexe et acyclique. En particulier, il est sans circuits et tel que tout noeud est source d'un nombre quelconque d'arcs mais cible d'au plus un arc.
  On y distingue le noeud *racine*, qui n'est cible d'aucun arc et les noeuds *feuilles*, qui ne sont source d'aucun arc. Un noeud à la fois racine et feuille est dit *isolé* (il forme un arbre à lui tout seul). Entre le noeud racine et tout autre noeud il existe un et un seul chemin. S'il existe un arc de n1 vers n2, n1 est dit *parent* de n2 et ce dernier est dit *enfant* de n1. S'il existe un chemin de n1 vers n2, n1 est dit *ancêtre* ou *ascendant* de n2, et ce dernier est dit *descendant* de n1. Un noeud d'un arbre ne peut évidemment pas être son propre ancêtre. Les noeuds de même parent sont dits *frères*.
  Un graphe orienté dont chaque composante connexe est un arbre est appelé *forêt* ou graphe arborescent.
  Le *niveau d'un noeud* quelconque d'un arbre est égal à 1 + la longueur du chemin qui part de la racine et qui aboutit à ce noeud. Le *niveau* de la racine est de 1.
  La *hauteur d'un arbre* est la longueur du plus long de tous les chemins qui partent de la racine et qui aboutissent à une feuille.

2. En anglais, directed acyclic graphs (DAG) ou acyclic digraphs.
3. Contrairement aux arbres orientés, la racine ne s'impose pas par ses propriétés, on la choisit arbitrairement.

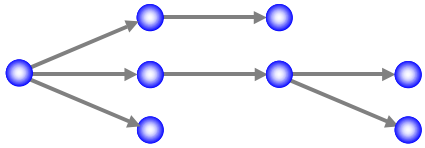


Figure A23.6.4 - Arbre orienté

- **Hiérarchie**

  Un graphe orienté est une *hiérarchie* s'il est connexe et sans circuits. Un noeud peut être cible et source d'un nombre quelconque d'arcs. On y distingue les noeuds *racines*, qui ne sont cible d'aucun arc et les noeud *feuilles*, qui ne sont source d'aucun arc. Un noeud à la fois racine et feuille est dit *isolé* (il forme la hiérarchie à lui tout seul). Plusieurs chemins peuvent exister entre deux noeuds quelconques. Par extension, nous parlerons encore des *enfants* d'un noeud ainsi que des *parents* d'un noeud. Un arbre est une hiérarchie particulière.

  Un graphe orienté dont chaque composante connexe est une hiérarchie est appelé *graphe hiérarchique*.

  Le *niveau d'un noeud* quelconque d'une hiérarchie est égal à 1 + la longueur du plus long chemin qui part d'une racine et qui aboutit à ce noeud. Le *niveau* d'une racine est de 1.

  La *hauteur d'une hiérarchie* est la longueur du plus long de tous les chemins qui partent d'une racine et qui aboutissent à une feuille.

  Une hiérarchie est parfois appelé *graphe décomposable en niveaux*.

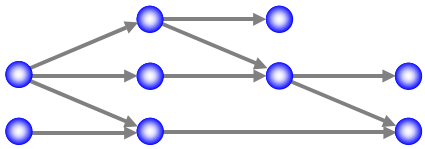


Figure A23.6.5 - Hiérarchie à quatre niveaux et deux racines

- **Graphe linéaire**

  Un *graphe linéaire* est un arbre orienté doté d'une seule feuille[4]. Il correspond à une liste de noeuds

  .

Figure A23.6.6 - Graphe linéaire

4. On limite ce concept aux graphes orientés, en effet, un graphe linéaire non orienté d'au moins 3 noeuds peut aussi être perçu de plusieurs façons comme un arbre doté de deux feuilles, ce qui complique la définition.

- **Treillis**
  Un *treillis* est une hiérarchie dotée d'une seule racine et d'une seule feuille. Un semi-treillis satisfait une seule de ces deux conditions.

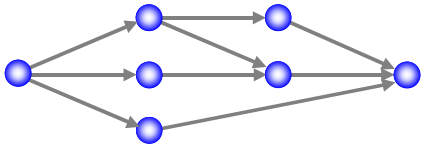


Figure A23.6.7 - Treillis

- **Graphe biparti**
  Les noeuds d'un *graphe biparti* sont répartis en deux catégories. Chaque arc relie un noeud d'une catégorie à un noeud de l'autre. Si le graphe est orienté, il est possible d'imposer qu'une catégorie joue le rôle d'origine et l'autre celui de cible.

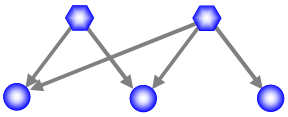


Figure A23.6.8 - Graphe biparti

- **Graphe attribué**
  On admet aisément que, dans une situation concrète, les noeuds soient dotés de propriétés spécifiques (coordonnées pour des lieux, prix unitaire pour des produits, numéro matricule et nom pour des agents). Des propriétés peuvent également être associées à des arcs : capacité d'une ligne téléphonique, débit moyen d'une canalisation, durée et longueur d'un trajet entre villes. Certaines propriétés peuvent être uniques parmi les arcs, de sorte qu'elles permettent de les identifier (numéro de voie, code de la ligne). D'autres définissent un type, et permettent si nécessaire de les classer (type de la ligne, nature de la canalisation, tension du câble, compagnie proposant des vols sur la ligne).

- **Multigraphe**
  Dans un *multigraphe*, plusieurs arcs peuvent être définis entre deux noeuds. Ces arcs doivent alors se distinguer par leurs propriétés. Par exemple, il existe plusieurs lignes aériennes entre deux destinations, mais elles sont proposées par des compagnies différentes.

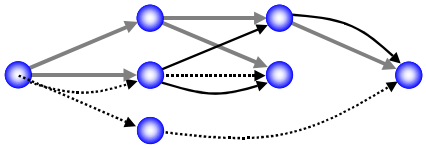


Figure A23.6.9 - Multigraphe à trois types d'arcs

A23.6.2Représentation des graphes orientés

Chaque arc est défini par un *couple* (n1,n2) où n1 est l'*origine* de l'arc et n2 sa *cible*. La représentation d'un tel graphe est immédiate. On peut en donner deux variantes équivalentes, selon que les arcs sont représentés par un type d'associations ou par un type d'entités (A23.6.10). L'asymétrie des arcs est indiquée par les rôles from (origine) et to (cible). L'identifiant du type d'entités ARC exprime qu'il ne peut exister plus d'un arc d'un noeud vers un autre. Dans cet exposé, et sauf mention contraire, nous privilégierons la première forme.

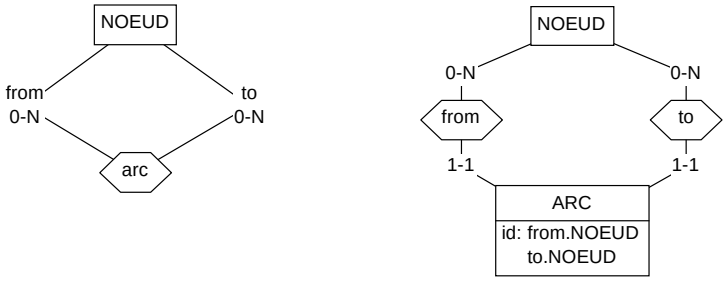


Figure A23.6.10 - Deux représentations équivalentes d'un graphe orienté

Dans une application spécifique, les types d'entités NOEUD et arc recevront des noms adéquats, tels que LOCALITE ou CARREFOUR pour NOEUD et ROUTE ou LIGNE pour arc. Dans ce contexte, les noms de rôles ou de types d'associations from et to devront aussi être adaptés, ce qui fera perdre au schéma l'information sur la direction des arcs. Afin de préciser de manière générique les extrémités origine et cible, nous utiliserons les stéréotypes from et to, ce qui nous laissera toute liberté d'utiliser des noms spécifiques, tels que origine et cible dans la A23.6.11.

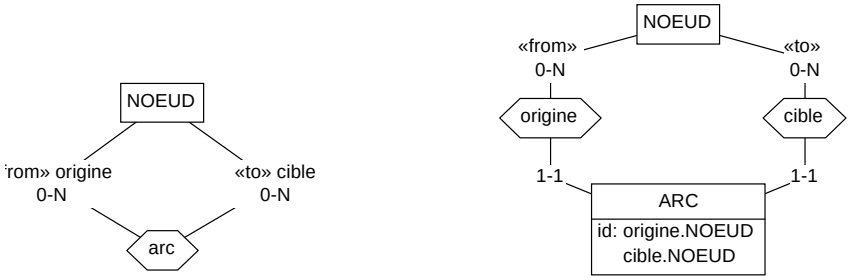


Figure A23.6.11 - Spécification des extrémités des arcs par les stéréotypes *from* et *to*

A23.6.3Représentation des graphes non orientés

Paradoxalement, la représentation des arcs non orientés s'avère plus complexe que celle des graphes orientés. Un arc d'un graphe non orienté est défini par un *ensemble* de deux noeuds {n1,n2}. On peut proposer la représentation de la A23.6.12 (gauche).

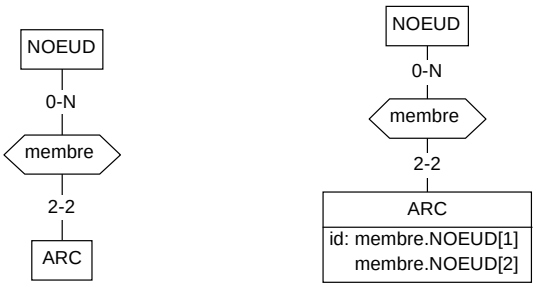


Figure A23.6.12 - Une première représentation d'un graphe non orienté

Or, il ne peut exister plus d'un arc entre deux noeuds quelconques, ce que le schéma de gauche n'exprime pas, puisque rien n'empêche que deux arcs soient formés de la même paire de noeuds. En outre, il n'autorise pas les arcs entre un noeud et lui-même. On proposera alors le schéma de droite, qui a cependant deux défauts : il utilise une forme non standard d'identifiant et il suggère un ordre sur les noeuds de l'arc. En outre, cet ordre exigerait qu'un second identifiant soit défini sur la permutation des composants.

Les schémas de la A23.6.13 proposent des variantes de représentation de graphe non orientés.

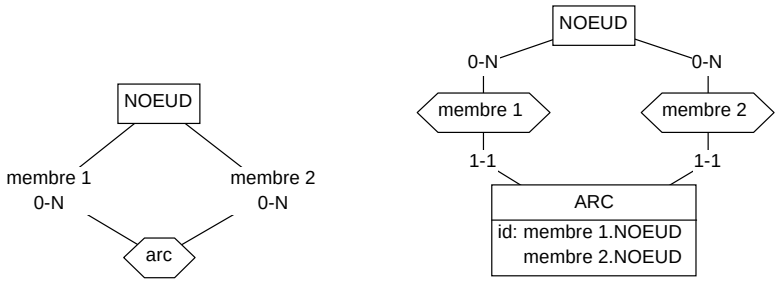


Figure A23.6.13 - Variantes de représentation de graphes non orientés

Ils utilisent cette fois des identifiants standard mais attribuent à chaque extrémité d'un arc un rôle déterminé, dénommé respectivement membre 1 et membre 2, comme si les arcs étaient orientés. Bien sûr, nous décidons d'ignorer ces rôles, en considérant que l'un vaut l'autre. Le problème n'est cependant pas aussi simple.

Soit {n1,n2} un arc du graphe. Nous le représentons selon le schéma de gauche par le couple (membre 1: n1, membre 2: n2) du type d'associations arc. Cependant, le couple (membre 1:n2, membre 2: n1) représente aussi cet arc. D'où la question : un arc {n1,n2} est-il représenté

1. par le couple (n1,n2), le couple (n2,n1) étant absent
2. par le couple (n2,n1), le couple (n1,n2) étant absent
3. ou par les deux couples (n1,n2) et (n2,n1) simultanément présents ?

Dans les cas 1 et 2, l'arc {n1,n2} existe si soit le couple (n1,n2) soit le couple (n2,n1) existent dans arc. Dans le cas 3, cet arc existe simplement si le couple (n1,n2) existe dans arc. Dans les premiers cas, il n'y a pas de redondance, mais la consultation des données sera plus complexe (un noeud intervient dans un arc s'il y est membre 1 **ou** membre 2), et dans le dernier cas, la redondance permet une consultation simple (un noeud intervient dans un arc s'il y est membre 1). Ces trois modes de représentation sont assortis chacun d'une contrainte d'intégrité régissant l'absence ou la présence du couple inverse.

Nous pouvons aussi convenir de conventions particulières telles que la suivante, applicable si les noeuds ont une propriété P identifiante (par exemple un code unique qui permet de les distinguer) :

l'arc {ni,nj} est représenté par le couple (ni,nj) de arc tel que $P(ni) \leq P(nj)$.

Ainsi, le membre 1 d'un arc est le noeud dont la propriété P est minimum. On peut de la sorte à la fois éviter la redondance et simplifier (raisonnablement) la consultation.

Il est également possible de laisser de côté cette question en indiquant, par le stéréotype symmetry la nature symétrique de la relation représentée par ces arcs (A23.6.14). On ne représente que l'un des deux couples symétriques. Dans ce schéma, on n'a pas assigné de nom aux deux rôles[5].

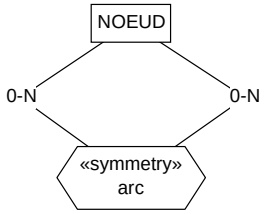


Figure A23.6.14 - Indication via le stéréotype "*symmetry*" de la nature symétrique de la relation correspondant aux arcs

A23.6.4Représentation des arbres et des forêts

Nous ne discuterons pas séparément ces deux types de graphes, que seul le nombre de racines distingue. Nous limiterons l'études aux graphes orientés. Un noeud d'un arbre est origine d'un nombre quelconque d'arcs mais est cible d'au plus un arc. La relation représentée par A est une fonction des noeuds cibles vers les noeuds origines (en d'autres termes, l'inverse de A est une fonction, ou A est l'inverse d'une fonction). Une première représentation est proposée à la A23.6.15 (gauche). Ce schéma semble violer la règle qui veut que les rôles d'un type d'associations aient des noms distincts. On pourrait dans ce cas spécifique considérer que les stéréotypes tiennent lieu de nom, ou attribuer des noms spécifiques tels que parent/père et enfant/fils (schéma de droite), eu égard aux rôles que jouent les noeuds dans une structure d'arbre.

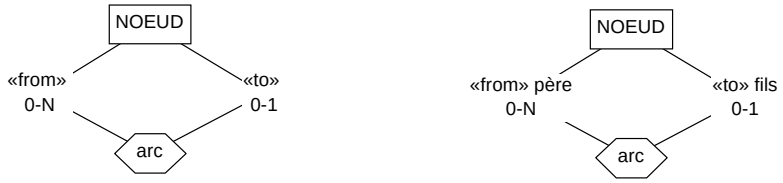


Figure A23.6.15 - Première représentation d'une forêt

Ce schéma est cependant trop général. Un arbre correspond en effet à une structure acyclique. Or, le schéma ci-dessus n'interdit pas les cycles. Comme nous l'avons observé à la section figure F.15.12.4, on ne peut imposer la propriété d'acyclicité par les constructions classiques du modèle Entité-association, de sorte que nous devons imaginer des palliatifs. Proposons ici, bien qu'il ait été montré qu'elle ne résout pas toutes les situations, une représentation via le stéréotype acyclic, assigné au type d'associations arc ou au type d'entités ARC selon le cas (A23.6.16).

5. Cette convention, qui viole la règle stipulant que les rôles d'un type d'associations portent des noms distincts, est rendue possible grâce à une particularité de l'affichage de l'atelier de l'atelier DB-MAIN.

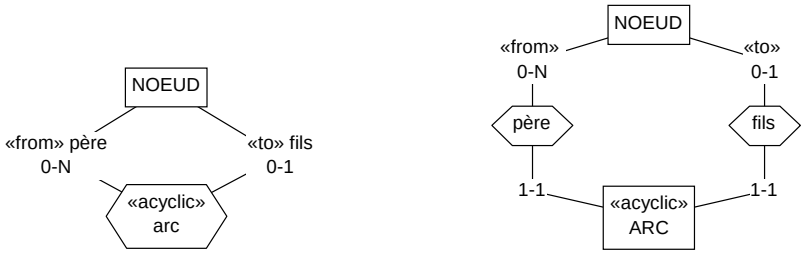


Figure A23.6.16 - Un arbre est une structure *acyclique*

L'existence de noeuds particuliers que sont les racines et les feuilles peuvent nous inciter à les représenter explicitement dans le schéma. Distinguons d'abord les racines des noeuds non racines. Les racines ne sont jamais cibles d'un arc alors que les noeuds non racines le sont obligatoirement. On précise ces propriétés dans le schéma de la A23.6.17. Notons que dans cette représentation les noeuds isolés sont considérés comme des racines.

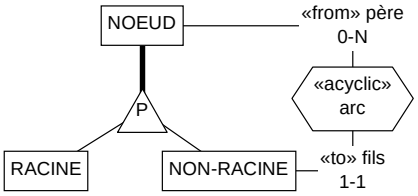


Figure A23.6.17 - Un arbre est constitué d'une racine et éventuellement de noeuds non racines, les seuls qui sont cibles d'arcs

Etendons la classification aux noeuds feuilles, qui ne peuvent être origine d'un arc. Nous partitionnons ainsi les noeuds en *racines*, *internes* et *feuilles* (A23.6.18). Tout arc a pour origine un noeud racine ou interne et a pour cible un noeud interne ou feuille. Remarquons qu'il faut encore distinguer les noeuds *isolés*, qui sont à la fois racine et feuille, mais qui ne rentrent pas dans la partition précédente, où les racines et les feuilles sont considérées comme des noeuds non isolés. De ce fait, tout noeud racine ou interne est obligatoirement origine d'arcs et tout noeud interne ou feuille est obligatoirement cible d'un arc. Cette représentation distingue clairement des racines et les feuilles, mais fait appel à une structure complexe, le type d'associations à rôles multi-TE.

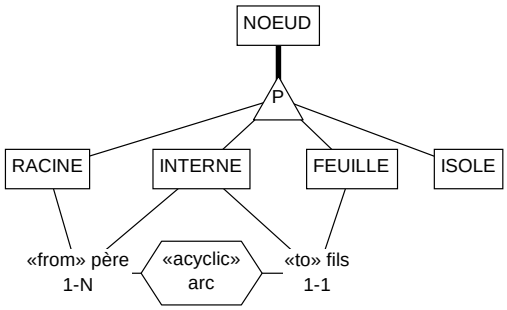


Figure A23.6.18 - Un arbre et ses quatre types de noeuds.

Une autre manière de classer les noeuds consiste à considérer, outre les noeuds isolés, les noeuds non racines (cibles d'au moins un arc) et les noeuds non feuilles (origine d'au moins un arc). Ces deux dernières catégories n'étant pas disjointes, nous devons construire une classification à deux niveaux (A23.6.19). Notons que les noeuds non-feuilles sont des noeuds pères et que les non-racines sont des fils. On envisagera d'adapter cette terminologie dans des situations concrètes. Par exemple, dans un organigramme d'entreprise, on utilisera le nom RESPONSABLE pour NON-FEUILLE et SUBORDONNE pour NON-RACINE.

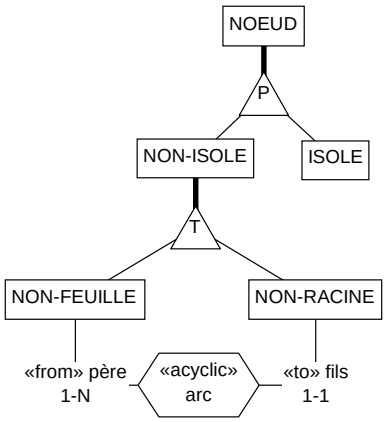


Figure A23.6.19 - Dans un arbre, un arc est composé d'un noeud non feuille et d'un noeud non racine.

S'il est nécessaire de distinguer les noeuds racines et les noeuds feuilles, on affinera la classification comme indiqué à la A23.6.20. On notera cependant qu'en toute rigueur, ce dernier schéma doit encore être complété par la contrainte suivante, qui malgré les apparences n'en est pas déductible[6] :

INTERNE = NON-FEUILLE ∩ NON-RACINE

ou, de manière équivalente :

RACINE ∩ FEUILLE = ∅

Le choix de la représentation adéquate dépendra des propriétés spécifiques de chaque catégorie de noeuds.

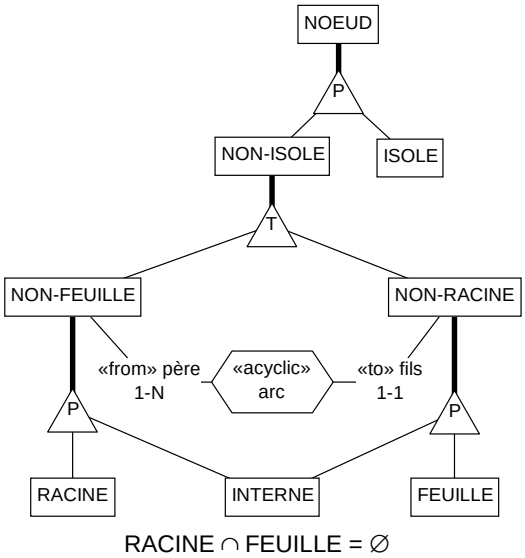


Figure A23.6.20 - Représentation complètes des différents types de noeuds d'un arbre

A23.6.5Représentation des graphes linéaires

Un graphe linéaire est un arbre dans lequel un noeud n'est origine que d'un seul arc, ou encore, un noeud n'a pas plus d'un enfant. Il suffit d'ajuster la cardinalité du rôle père de [0-N] en [0-1], ou de [1-N] en [1-1], selon le schéma. Il reste nécessaire d'imposer l'acyclicité du graphe, à défaut de quoi il serait possible de construire des boucles. La A23.6.21 donne deux variantes de schémas d'arbres adaptées aux graphes linéaires.

A23.6.6Représentation des graphes hiérarchiques

Un graphe hiérarchique présente des similitudes avec les arbres. La principale différence réside dans le fait qu'un noeud peut avoir plus d'un parent. Nous reprendrons

6. INTERNE n'est qu'une partie de l'intersection de NON-FEUILLE et NON-RACINE. En particulier, rien n'empêche selon ce schéma qu'un noeud racine appartienne au type NON-RACINE !

donc les représentations des arbres et des forêts, dans lesquelles nous changeons la cardinalité du rôle fils de [0-1] en [0-N], ou de [1-1] en [1-N], selon le schéma. Bien entendu, on déclarera les graphes hiérarchiques *acycliques*. La A23.6.22 donne deux variantes de schémas de représentation de graphes hiérarchiques.

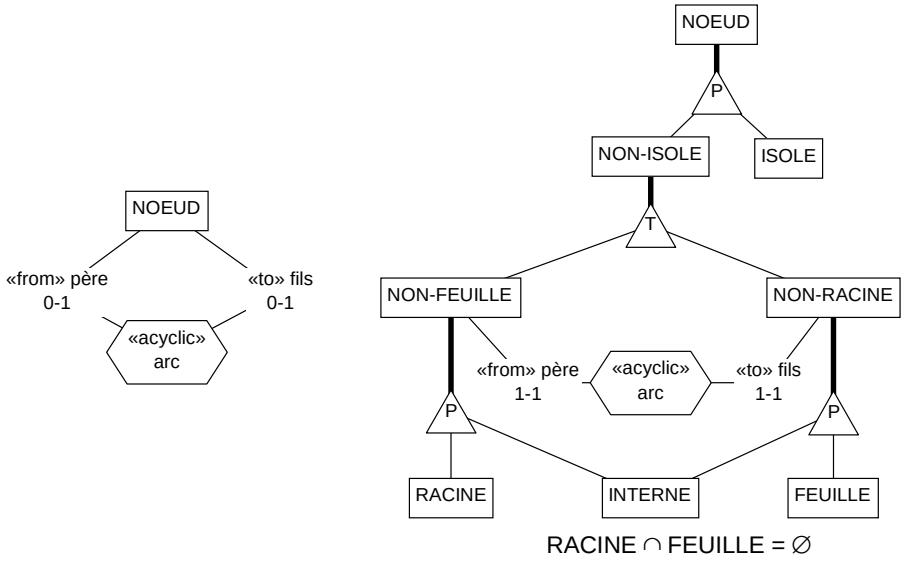


Figure A23.6.21 - Deux représentations de graphes linéaires

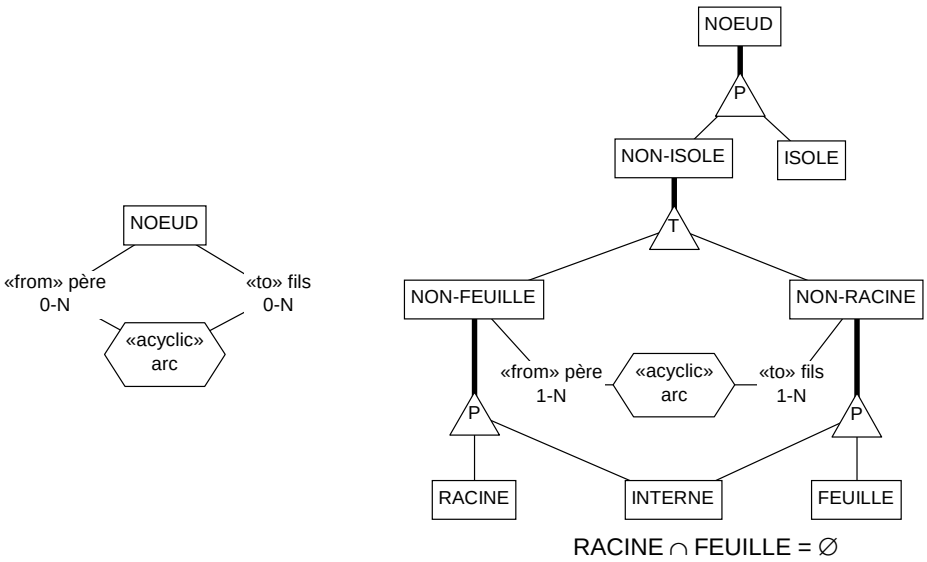


Figure A23.6.22 - Deux représentations de graphes hiérarchiques

A23.6.7Représentation des graphes bipartis

Considérons un graphe constitué de deux types de noeuds, dénommés NOEUD1 et NOEUD2, et d'arcs orientés de noeuds du premier type vers des noeuds du second type. Il s'agit d'un graphe hiérarchique particulier, dont les racines appartiennent au type NOEUD1 et les feuilles au type NOEUD2, et dont la hauteur maximum est de 1 (A23.6.23). Le stéréotype acyclic est ici inutile.

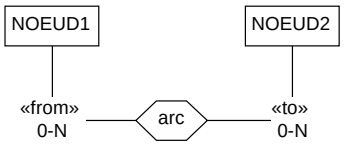


Figure A23.6.23 - Graphe biparti hiérarchique

A23.6.8Représentation des graphes attribués

Nous discuterons de cette question sur des graphes hiérarchiques; elle est cependant généralisable à tous les types de graphes. Pour simplifier les schémas, nous ne ferons pas apparaître les stéréotypes from, to et acyclic que le lecteur ajoutera lui-même s'il l'estime nécessaire.

En pratique, on associe aux noeuds des propriétés, dont une au moins est identifiante. Il est en effet généralement exigé de pouvoir désigner univoquement chaque noeud du graphe. Il est aussi fréquent que les noeuds soient typés en fonction de leur position dans le graphe, et que selon leur type, ils possèdent des propriétés spécifiques. Considérons par exemple le cas d'une nomenclature de produits, dans laquelle on indique pour chaque produit *composé* les *composants* qui interviennent dans sa fabrication. Une telle nomenclature s'exprime le plus généralement[7] sous la forme d'un graphe hiérarchique dont les racines correspondent aux *produits finis*, les feuilles aux *matières premières* et les noeuds internes aux *produits semi-finis*. Il est évident que chacune de ces catégories possède des caractéristiques propres.

La A23.6.24 représente un graphe abstrait dont les noeuds sont dotés de propriétés. L'une d'elles est traduite sous la forme d'un attribut identifiant (IdNoeud). En toute généralité, ces propriétés s'expriment sous la forme d'attributs, de rôles joués dans des types d'associations et de contraintes d'intégrité. Lorsque les noeuds d'un graphe orienté sont susceptibles d'être dotés de propriétés fonction de leur position dans le graphe, on reprendra l'une des représentations qui identifie clairement ces positions pour y distribuer les propriétés (voir par exemple le schéma de droite de la A23.6.22).

7. Cette réserve tient par exemple compte des situations dans lesquelles un produit intervient dans sa propre fabrication (*groisil* dans l'industrie verrière, (recyclage de déchets, levain dans la fabrication de la pâte à pain), ce qui peut rendre le graphe non hiérarchique.

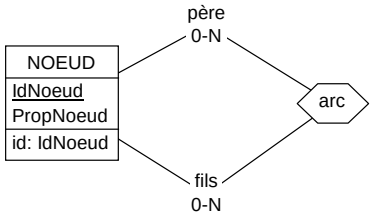


Figure A23.6.24 - Type de noeuds doté d'attributs dont l'un est identifiant

Les arcs eux-mêmes peuvent se voir dotés de propriétés traduites sous la forme d'attributs ou de rôles additionnels, pris par d'autres types d'entités, qui peuvent d'ailleurs être eux-mêmes des noeuds du graphe (A23.6.25).

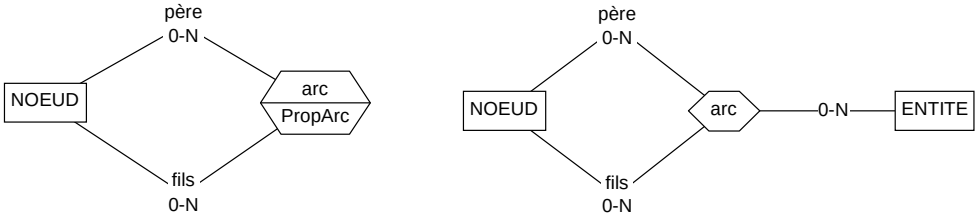


Figure A23.6.25 - Les arcs peuvent être dotés d'attributs ou de rôles additionnels

Dans un graphe hiérarchique, lorsque des propriétés sont représentées par de nouveaux rôles, on indiquera par les stéréotypes from et to les deux rôles qui définissent la relation binaire hiérarchique (A23.6.26).

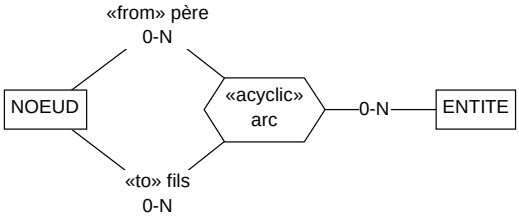


Figure A23.6.26 - Arcs binaires avec propriétés représentés par des types d'associations n-aires

Les propriétés peuvent servir à constituer des identifiants des arcs. La A23.6.27 montre un schéma dans lequel les arcs sont dotés d'un attribut IdArc qui permet de les distinguer. Si le graphe obéit aux règles classiques, il n'existe pas plus d'un arc entre deux noeuds, de sorte qu'il est nécessaire de préciser explicitement cette contrainte par un identifiant secondaire[8]. La relaxation de cette contrainte conduirait aux multigraphes, qui seront étudiés plus loin.

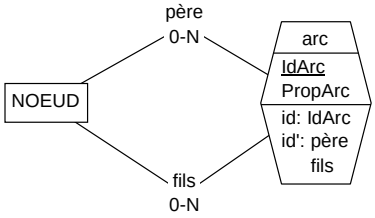


Figure A23.6.27 - Identifiants des arcs

Il n'est pas rare qu'un attribut soit utilisé pour identifier un arc parmi tous les arcs issus du même noeud père, ou aboutissant au même noeud fils. Si cet attribut est un entier, il peut servir à définir une *relation d'ordre* sur ces arcs, et par conséquent sur les fils de chaque noeud père (A23.6.28), ou les pères de chaque noeud fils. On retrouve une variante des identifiants cycliques étudiés à la section figure F.15.10.2. Ici encore, il est nécessaire de préciser par un autre identifiant l'unicité des arcs entre deux noeuds.

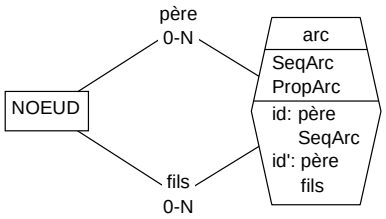


Figure A23.6.28 - Identifiant relatif et relation d'ordre sur les arcs

Lorsque le graphe se réduit à une *forêt* (A23.6.29, gauche), les propriétés des arcs, qu'il s'agisse d'attributs ou de rôles, peuvent migrer vers les noeuds fils. Il faut cependant être attentif au fait que seuls les noeuds fils (donc non racines) peuvent hériter de ces propriétés. Celles-ci deviennent par conséquent facultatives et sont soumises à une contrainte de coexistence (A23.6.29, centre). On peut éviter ces propriétés facultatives en mettant en évidence le concept de noeud fils (A23.6.29, droite). Les propriétés constituant des identifiants se traitent de la même manière. Le schéma de la A23.6.28 s'exprimera comme indiqué à la A23.6.30. On notera que l'identifiant secondaire a disparu, en raison des propriétés particulières (fonctionnelles) des arbres.

8. Rappelons les règles relatifs aux identifiants des types d'associations : (1) tout rôle de cardinalité [0-1] ou [1-1] est un identifiant implicite, (2) en l'absence d'identifiants implicites et d'identifiants déclarés, l'ensemble des rôles forme l'identifiant implicite.

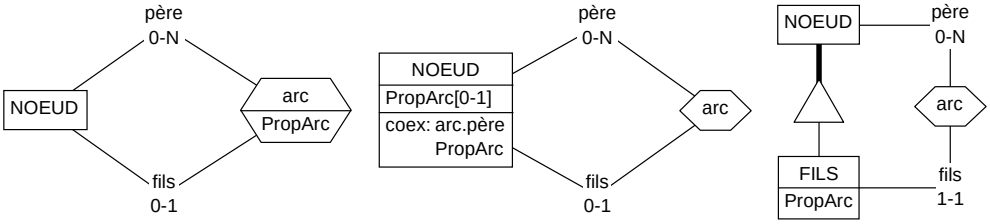


Figure A23.6.29 - Migration des attributs d'arcs (gauche) vers les noeuds fils (centre, droite) dans les forêts

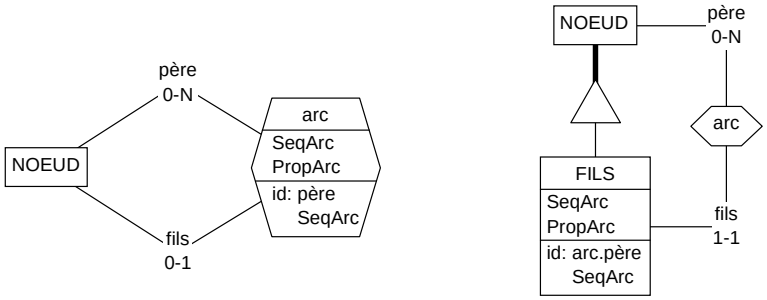


Figure A23.6.30 - Migration d'un identifiant relatif d'arcs (gauche) vers les noeuds fils (droite) dans les forêts

A23.6.9Représentation des multigraphes

Dans un multigraphe, deux noeuds peuvent être reliés par plus d'un arc. Par exemple, dans un graphe hiérarchique, un noeud peut être connecté à un fils selon un arc, et à ce même fils selon un autre arc. Il est clair que dans ce type de graphe la seule connaissance des deux extrémités d'un arc ne suffit plus à identifier celui-ci. Dans un multigraphe, les arcs sont forcément dotés de propriétés permettant, au moins, de distinguer les arcs de mêmes extrémités.

Observons aussi qu'un multigraphe ne peut être une forêt, et qu'il est, au moins, hiérarchique. L'exemple de la A23.6.31 est celui d'un multigraphe hiérarchique qui admet plusieurs arcs entre deux noeuds, à condition qu'ils soient de type (attribut TypeArc) différents. Ici encore, les stéréotypes from, to et acyclic ont été ignorés et seront introduits si nécessaire.

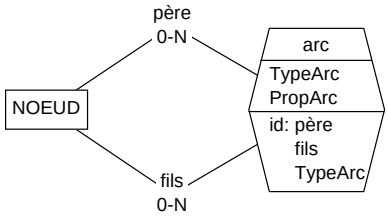


Figure A23.6.31 - Multigraphe dans lequel les arcs de mêmes extrémités se distinguent selon la valeur de l'attribut TypeArc

A23.6.10 Quelques applications

Nous présenterons ci-après quelques exemples d'application des structures de graphes étudiées dans cette section.

a) Organigramme

On désire représenter les relations hiérarchiques inter-personnelles dans une organisation :

> *Une personne peut être subordonnée à une autre, qui est son responsable. Nul n'est son propre responsable, ni directement ni indirectement.*

Le graphe de ces relations est une forêt. On en propose deux représentations équivalentes à la A23.6.32. Dans la seconde, les personnes non subordonnées ont été dénommées *directeurs*.

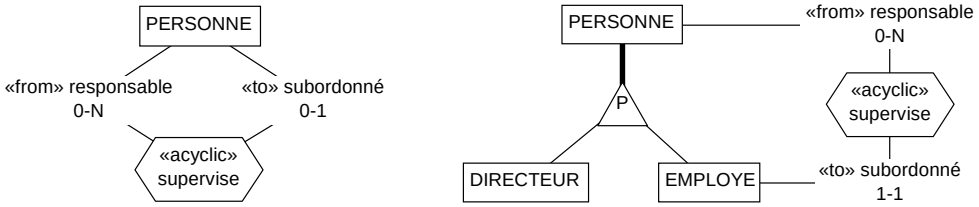


Figure A23.6.32 - Représentation des relations hiérarchiques entre personnes

b) Relations entre collègues

> *Dans le cadre de chacune de ses activités professionnelle, toute personne développe des relations avec ses collègues. Toute personne est le collègue de ses collègues.*

Ces relations correspondent à un graphe non orienté. On choisi une représentation orientée dans laquelle on associe à chaque personne (le sujet) l'ensemble de ses collègues (collègue). L'indication de symétrie (stéréotype symmetry) est à interpréter comme suit : si le couple (p1,p2) se trouve dans collègue, alors le couple (p2,p1) s'y

trouve aussi. La question de savoir si une personne est son propre collègue reste ouverte[9] . . .

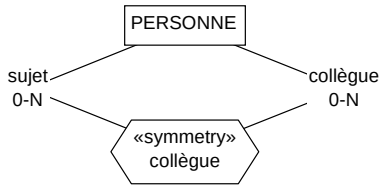


Figure A23.6.33 - Relations entre collègues

c) Mères et filles

Cet énoncé est très court :

Une mère donne à ses filles des prénoms distincts.

La structure mère/fille est de toute évidence une forêt[10], mais qui est soumise à une contrainte particulière : *il est possible d'identifier une personne de sexe féminin par son prénom et sa mère*. C'est ce qu'exprime le schéma de la A23.6.34, qui comporte un identifiant cyclique (section figure F.15.10.2). L'identifiant qui traduit la contrainte a été déclaré secondaire, bien que ses composants soient obligatoires pour FILLE. En effet, si FEMME devait recevoir un identifiant primaire, ce qui serait naturel, vu que ce type d'entités ne dispose d'aucun moyen d'identification propre (une *femme* non répertoriée comme *fille* n'est pas identifiable), cet identifiant serait également primaire pour FILLE.

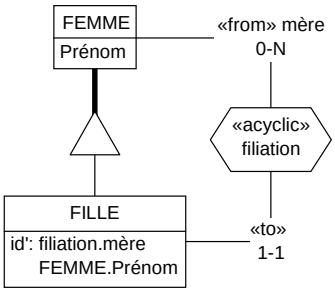


Figure A23.6.34 - Identifiant hybride dans une forêt : *mères et filles*

d) Développement d'arbres fruitiers

Cet exemple, bien que relevant d'un domaine tout différent, présente la même structure que le précédent. Il m'a été suggéré, il y a bien des années, par J.J. Claustriaux, professeur à la Faculté des sciences agronomiques de Gembloux.

9. En d'autres termes, la relation représentée par les arcs est-elle *réflexive* ?
10. Pour simplifier l'exemple, on se limite aux personnes d'un même sexe.

Il s'agit de représenter la structure et la croissante de certains arbres fruitiers (en l'occurrence des pommiers). D'une manière générale, et en simplifiant à outrance, un arbre est formé d'axes (branches) dotés chacun d'une séquence de noeuds. Chaque axe (sauf l'axe principal, qui est le tronc) pousse au départ d'un noeud d'un axe plus ancien. La structure qui résulte de ces règles est, sans surprise, celle d'un arbre. On peut désigner univoquement un axe en précisant son axe père et le numéro du noeud dont il est issu sur cet axe.

Le schéma décrivant cette structure est représenté à la A23.6.35.

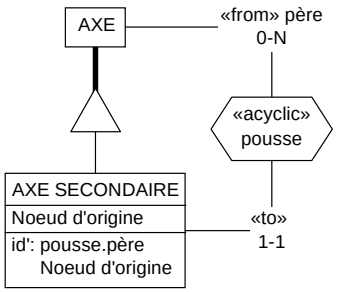


Figure A23.6.35 - Identifiant hybride dans une forêt : *structure d'arbres fruitiers*

Le lecteur est invité à proposer un schéma conceptuel plus complet, construit à partir de l'énoncé ci-dessous.

Les pommiers se développent selon le schéma suivant : chaque pommier est constitué d'un certain nombre d'axes. A l'exception du premier, qui est le tronc et qui est déclaré d'ordre 0, chaque axe est issu d'un noeud de son axe père, avec lequel il fait un certain angle dans le plan vertical. Son ordre est celui de son père + 1. Chaque axe possède un certain nombre de noeuds qui se distinguent par leur distance à partir de l'origine de l'axe. Les noeuds d'un axe portent un numéro d'ordre selon leur éloignement de l'origine. Chaque année, la croissance s'effectue selon deux modes, par prolongement ou par ramification. Le prolongement ajoute un segment à un axe, tandis que la ramification ajoute un axe d'ordre supérieur à partir d'un noeud. On admet que chaque pommier est identifié par un code.

e) Méta-schéma pour les bases de données HP Image 3000

Les schémas de bases de données *Image 3000 de HP*[11] constituent des exemples remarquables de *multigraphes hiérarchiques bipartis* (A23.6.36).

11. Ainsi que des bases de données TOTAL, anciennement de Cincom, et actuellement propriété de Computer Associates. La rumeur veut qu'un employé de Cincom ait un jour quitté la société, une bande magnétique sous le bras, pour se faire embaucher par HP, lequel, peu de temps après, à sorti un SGBD présentant une forte ressemblance avec TOTAL. Mais on dit tant de choses ...

Tout schéma Image est formé de Data Sets *(DS, ou types d'enregistrements) de deux types :* Master *et* Detail. *Des associations peuvent être spécifiées d'un* DS Master *vers un* DS Detail. *Un type d'associations (*relationship*) possède un nom unique. Plusieurs types d'associations peuvent être définis d'un même* DS Master *vers le même* DS Detail.

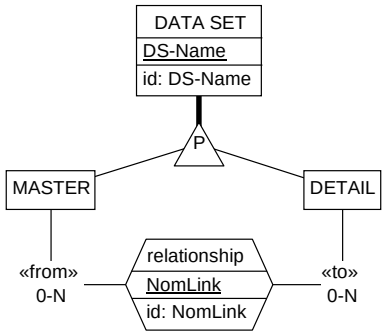


Figure A23.6.36 - Un exemple de *multigraphe hiérarchique biparti* : le métaschéma (partiel) des bases de données Image 3000 de HP

Malgré les limitations apparentes de ce modèle de données, qui semble n'autoriser que des graphes hiérarchiques à deux niveaux seulement, tout schéma Entité-association peut être traduit en schéma Image.

Deux remarques

- Les types d'associations IMAGE sont de classe fonctionnelle 1:N. N'y a-t-il pas contradiction avec les cardinalités du type d'associations relationship du schéma F.A23.6.36 ?
- Le type d'associations relationship a pour identifiant un attribut. Cette forme d'identifiant est-elle légitime ?

f) Gestion de projets (décomposition et ressources)

Tout projet est constitué de tâches. Une tâche est elle-même constituée de tâches plus élémentaires. Une tâche peut se retrouver dans plus d'une tâche de plus haut niveau. Pour les tâches élémentaires (non constituées d'autres tâches), on connaît la durée et les ressources nécessaires (nature et quantité).

g) Gestion de projets (ordonnancement)

Tout projet est constitué de tâches qui doivent s'exécuter dans un certain ordre. On précise par exemple qu'une tâche ne peut commencer que lorsque certaines tâches sont terminées. Pour chaque tâche on connaît la durée normale.

h) Plan routier d'une agglomération

Les rues d'une agglomération relient chacune deux point de jonction (places, carrefours, lieux-dits, etc.) Les rues portent des noms distincts. Certains points de jonction portent également un nom. Ces derniers sont distincts. Les points de jonction portent en outre un code identifiant. Pour chaque rue on connaît le (ou les) sens de circulation autorisé(s).

i) Représentation de grammaires

On considère la notation BNF, ou toute variante, permettant de spécifier une grammaire. Elaborer un schéma conceptuel de cette notation. *Observation* : toute instance de ce schéma est la grammaire d'un langage particulier exprimée dans cette notation.

j) Représentation de programmes

On considère un langage de programmation dont on connaît la grammaire. Elaborer un schéma conceptuel de la structure de la grammaire de ce langage. *Observation* : toute instance de ce schéma est un programme particulier exprimé dans ce langage.

A23.7 RÉTRO-INGÉNIERIE D'UNE BASE DE DONNÉES RELATIONNELLE

On considère une application utilisant une base de données relationnelle. On ne dispose plus que des fragments de code ci-après, sans documentation (figure A23.7.1). On se propose de reconstituer le schéma logique complet et un schéma conceptuel de la base de données. Les résultats obtenus devront encore être validés par consultation d'autres sources d'information.

A23.7.1 Extraction du schéma physique

L'analyse du code DDL relatif aux tables et aux index conduit au schéma physique brut de la figure A23.7.2. L'index *unique* sur COMMANDE est interprété comme un identifiant secondaire.

A23.7.2 Extraction du schéma logique

Le schéma physique est assez riche en constructions explicites puisqu'il contient des identifiants et des clés étrangères. Son analyse pourrait nous suggérer que PIECE.NOMP pourrait être une clé étrangère implicite vers PROJET. Cependant, au-delà de l'égalité des noms, les types de valeurs ne sont que faiblement compatibles, PROJET.NOMP étant plus long que PIECE.NOMP. On propose d'écarter la sugges-

tion. Il s'agit à présent de rechercher les constructions implicites que nous suggère l'analyse du code additionnel.

```
create table PROJET (
  NOMP char(20) not null,
  STAFF char(110) not null,
  primary key(NOMP));

create table EMPLOYE (
  NUME char(6) not null,
  NOM char(30) not null,
  ADRESSE varchar(60),
  primary key(NUME));

create table HISTORIQUE (
  CODE_E char(6) not null,
  STAT varchar(60) not null,
  DESCR varchar(200) not null,
  primary key(CODE_E)
  foreign key (CODE_E)
  references EMPLOYE);

create table TECHNICIEN (
  NUME char(6) not null,
  QUALIF char(10),
  DATE_E date not null,
  primary key(NUME),
  foreign key (NUME)
  references EMPLOYE);

create table PIECE (
  NUMP char(10) not null,
  NOMP varchar(25) not null,
  primary key(NUMP));
```

```
create table COMMANDE (
  DATEC date not null,
  DESTIN char(20) not null,
  RESP char(6) not null,
  OBJET char(10) not null,
  foreign key (DESTIN)
  references PROJET,
  foreign key (RESP)
  references EMPLOYE,
  foreign key (OBJET)
  references PIECE);

create unique index XPRO
  on PROJET(NOMP);
create unique index XEMP
  on EMPLOYE(NUME);
create unique index XHIS
  on HISTORIQUE(CODE_E);
create unique index XTECH
  on TECHNICIEN(NUME);
create unique index XCOM
  on COMMANDE(DATEC,DESTIN,
  RESP,OBJET);
create unique index XPIECE
  on PIECE(NUMP);

create view EMP_HIST(NE)      [1]
as select NUME from EMPLOYE
   where NUME not in
   (select CODE_E from HISTORIQUE);
  . . .
select 'Erreur : ', NE
       from EMP_HIST;         [2]
  . . .
```

```
  procedure DISPL-PROJET(NOM_P:char[20])                    [3]
  variable section                                          [4]
    MEMBRES :array[1:11] of                                 [5]
    record DATE_DEBUT : date;                               [6]
           NE : char(6);                                    [7]
    end-record;                                             [8]
    NOM : char(30);                                         [9]
    I : integer;                                           [10]
```

```
body section                                                          [11]
  . . .                                                               [12]
  select STAFF into :MEMBRES from PROJET                              [13]
  where NOMP = :NOM_P;                                                [14]
  if SQLCODE = 0 then                                                 [15]
    select NOM into :NOM from EMPLOYE                                 [16]
    where NUME = :MEMBRES[1].NE;                                      [17]
    display("Directeur : ",NOM);                                      [18]
    for I := 2 to 11 while MEMBRES[I].NE <> 0 do                      [19]
      select NOM into :NOM from EMPLOYE                               [20]
      where NUME = :MEMBRES[I].NE;                                    [21]
      display("Assistant : ",NOM);                                    [22]
    end-for                                                           [23]
  else                                                                [24]
    ...                                                               [25]
  end-if;                                                             [26]
end-procedure DISPL-PROJECT;                                          [27]
```

Figure 7.1 - Fragments de code disponibles

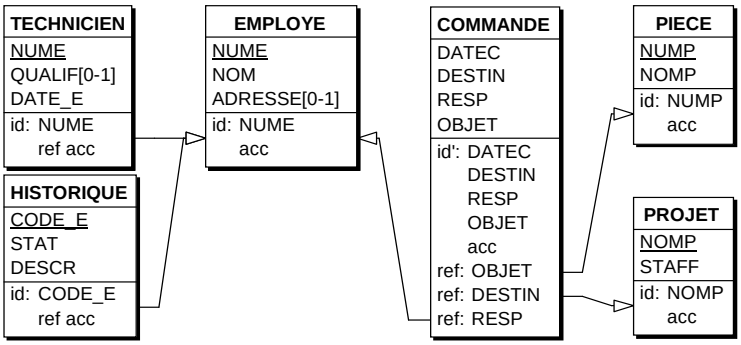


Figure 7.2 - Schéma physique brut

Examinons d'abord la vue EMP_HIST (ligne [1]). Elle repère les lignes d'EMPLOYE auxquelles ne correspond aucune ligne d'HISTORIQUE. Plus loin, une requête (ligne [2]) extrait les lignes de cette vue en qualifiant d'*Erreur* le résultat obtenu. *Conclusion* : pas de ligne d'EMPLOYE sans ligne d'HISTORIQUE. D'où la transformation de la contrainte ref d'HISTORIQUE en equ.

L'examen du module DISPL-PROJET nous apprend [13] que la valeur de STAFF est rangée dans la variable MEMBRES, structurée en un tableau de champs composés [5-8]. *Conclusion* : on assigne cette structure de tableau à la colonne STAFF (figure A23.7.3/gauche).

Les lignes [16-18] accèdent à une ligne d'EMPLOYE sur la base de la valeur de NE du 1er élément de la variable MEMBRES. Le résultat est qualifié de 'Directeur'. On observe qu'on n'envisage pas qu'il n'y en ait aucun.

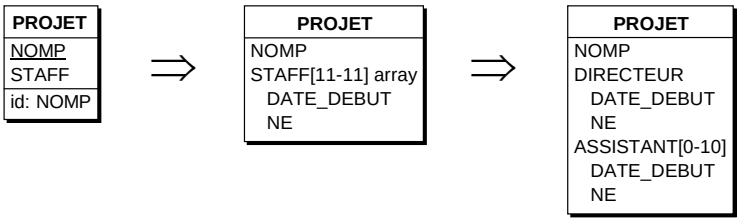


Figure 7.3 - Affinements successifs de la colonne STAFF

Cette première ligne représente le membre du staff qui joue le rôle de directeur du projet. On l'extrait du tableau sous la forme d'un champ composé nommé DIRECTEUR. Les lignes [19-23] fonctionnent de la même manière pour des membres du staff qualifiés d''Assistant'. Il y en a de 0 à 10, ce qu'on traduit par un champ de type tableau (figure A23.7.3/droite). On fait en outre l'hypothèse (qu'on vérifiera, par exemple par une analyse des données) que les valeurs de NE sont uniques pour une ligne de PROJET. On admet aussi que l'ordre des assistants est indifférent, de sorte que le tableau est simplement un ensemble). En outre, les composants NE sont traités comme des clés étrangères vers la table EMPLOYE.

Après avoir supprimé les index, on en déduit le schéma logique enrichi de la figure A23.7.4.

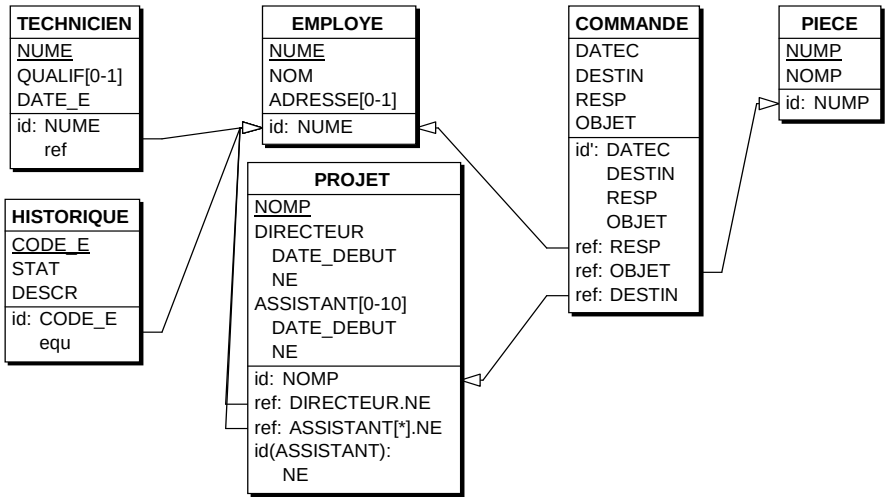


Figure 7.4 - Schéma logique complet

A23.7.3Conceptualisation

Nous opérerons en deux phases. La première consiste à dériver un schéma conceptuel brut, lequel sera ensuite normalisé.

PROJET contient deux attributs[12] complexes, DIRECTEUR et ASSISTANT, qui apparaissent comme des constructions optimisées permettant de minimiser le nombre de tables et donc le nombre de jointures. On extrait ces attributs sous la forme de types d'entités (figure A23.7.5).

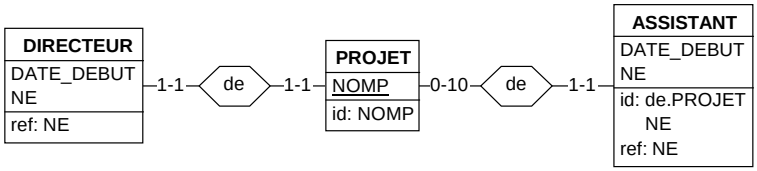


Figure 7.5 - Extraction des attributs complexes DIRECTEUR et ASSISTANT

Toutes les clés étrangères sont désormais au niveau 1 dans leurs parents. Nous pouvons donc les transformer en types d'associations fonctionnels. Le résultat correspond au schéma conceptuel brut (figure A23.7.6). Celui-ci contient plusieurs constructions qui peuvent être améliorées par normalisation.

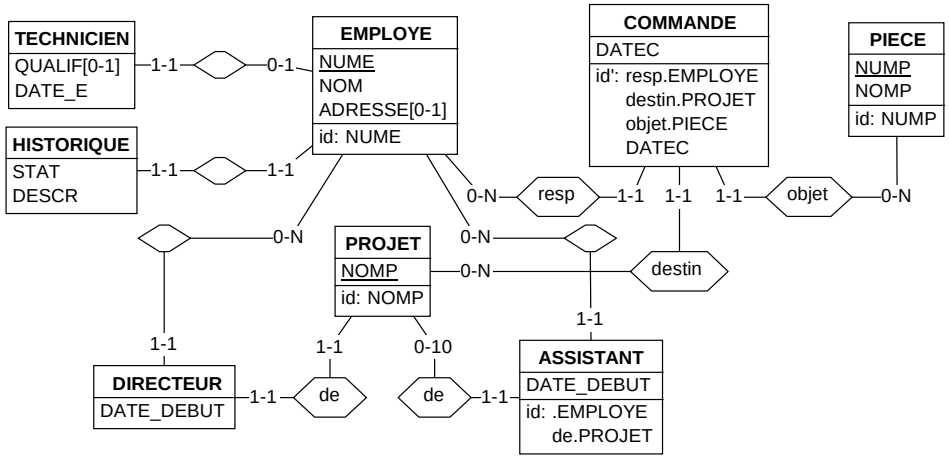


Figure 7.6 - Schéma conceptuel brut

- L'assemblage EMPLOYE/HISTORIQUE apparaît comme un *type d'entités fragmenté*. On fusionne ces types d'entités de manière à simplifier cette construction.
- Les entités représentées par TECHNICIEN forment une catégorie d'employés. On interprète donc le type d'associations 1:1 entre TECHNICIEN et EMPLOYE comme la représentation d'une relation *is-a*.

12. À ce stade, on parlera désormais de types d'entités et d'attributs.

- On donne à l'identifiant de COMMANDE le statut *primaire*.
- Le type d'entités COMMANDE apparaît comme un *type d'entités association*. On choisit de le transformer en un type d'association ternaire[13].
- On transforme enfin les noms de manière à les rendre plus expressifs.

On obtient alors le schéma conceptuel normalisé de la figure A23.7.5.

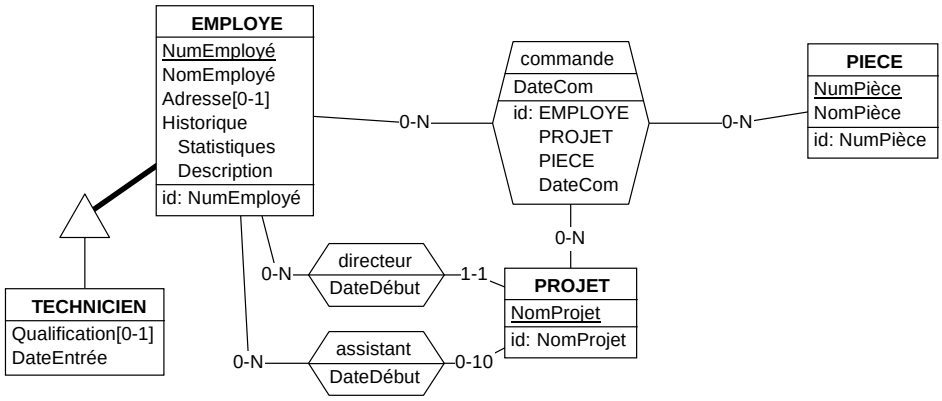


Figure 7.7 - Schéma conceptuel normalisé

A23.8GESTION DE CLÉS

Le schéma de la A23.8.1 est fourni par le responsable de la gestion des clés dans une entreprise. Il correspond à la structure *améliorée* des tables Access de l'application de gestion des clés. Proposer un schéma conceptuel de cette base de données.

13. Cette transformation est facultative, en fonction des standards locaux.

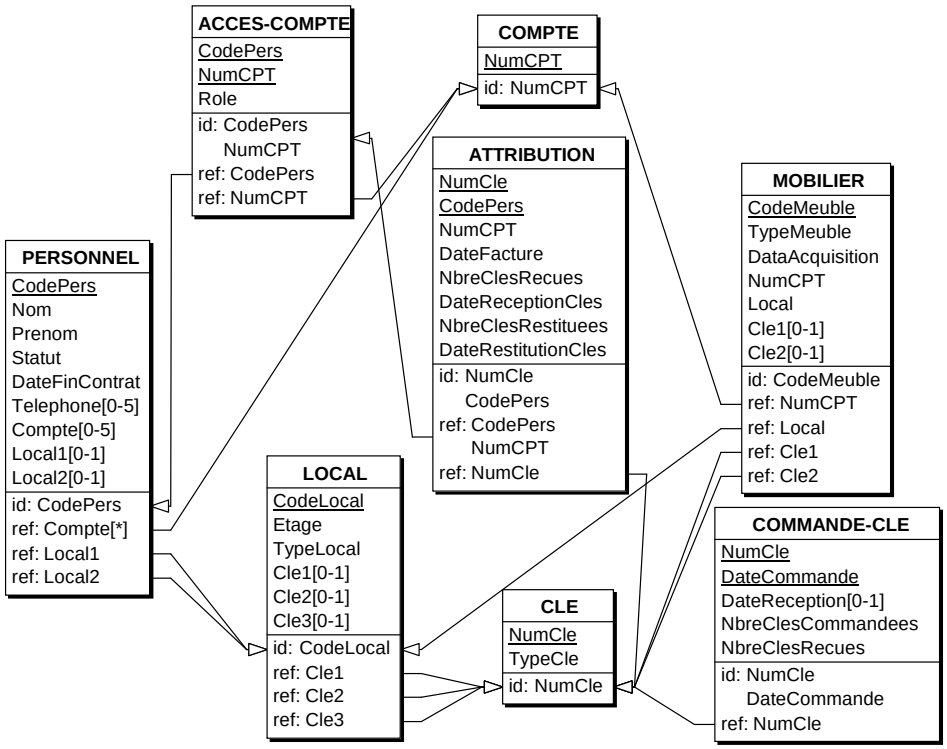


Figure A23.8.1 - Gestion des clés d'une entreprise. Schéma logique source.

Le schéma de départ est à considérer comme le schéma logique affiné de la base de données. En particulier, les colonnes Telephone et Compte ont été reconnues multi-valuées et chaque valeur de Compte a été identifiée comme une clé étrangère vers la table COMPTE. Il reste donc à conceptualiser ce schéma.

a) Les attributs sériels

Le schéma comporte trois groupes d'attributs sériels à transformer en attributs multi-valués :

- Local, Local1 de PERSONNEL traduisant l'attribut Local[0-2];
- Cle1, Cle2, Cle3 de LOCAL traduisant l'attribut Cle[0-3];
- Cle1, Cle2 de MOBILIER traduisant l'attribut Cle[0-2].

On obtient ainsi le schéma de la A23.8.2.

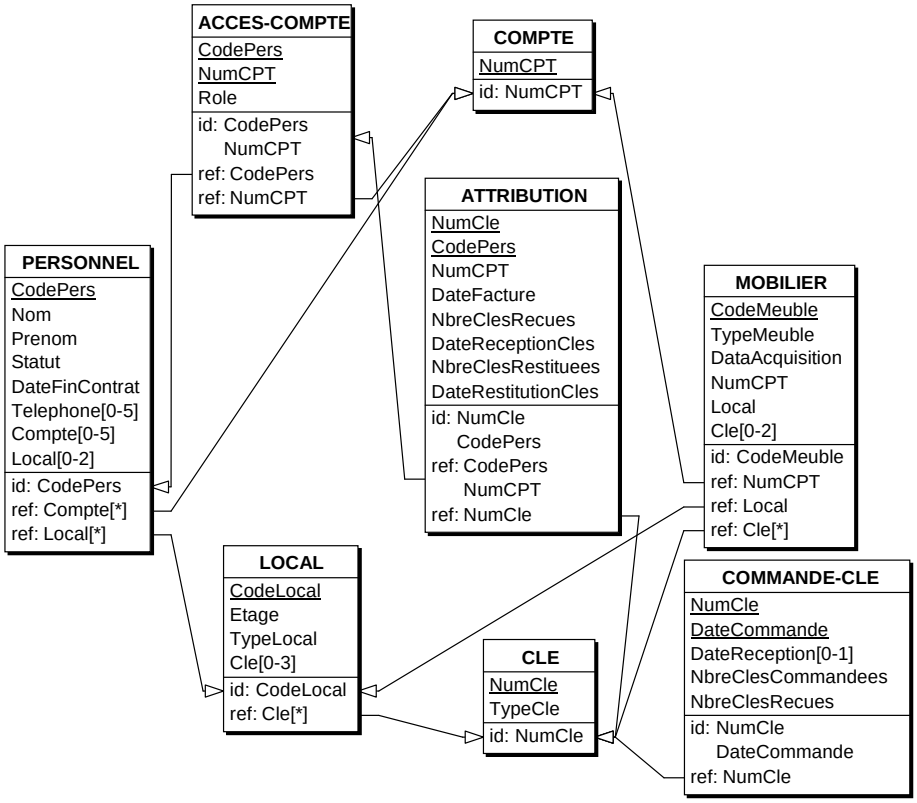


Figure A23.8.2 - Gestion des clés d'une entreprise.
Explicitation des attributs multivalués

b) Les clés étrangères

Les clés étrangères sont transformées en types d'associations. On peut ainsi traiter successivement :

- PERSONNEL.Compte transformé en accès;
- PERSONNEL.Local transformé en local;
- LOCAL.Cle transformé en cle;
- MOBILIER.NumCPT transformé en affecte;
- MOBILIER.Local transformé en local;
- MOBILIER.cle transformé en cle;
- COMMANDE-CLE.NumCle transformé en ref.

Le résultat est présenté à la A23.8.3.

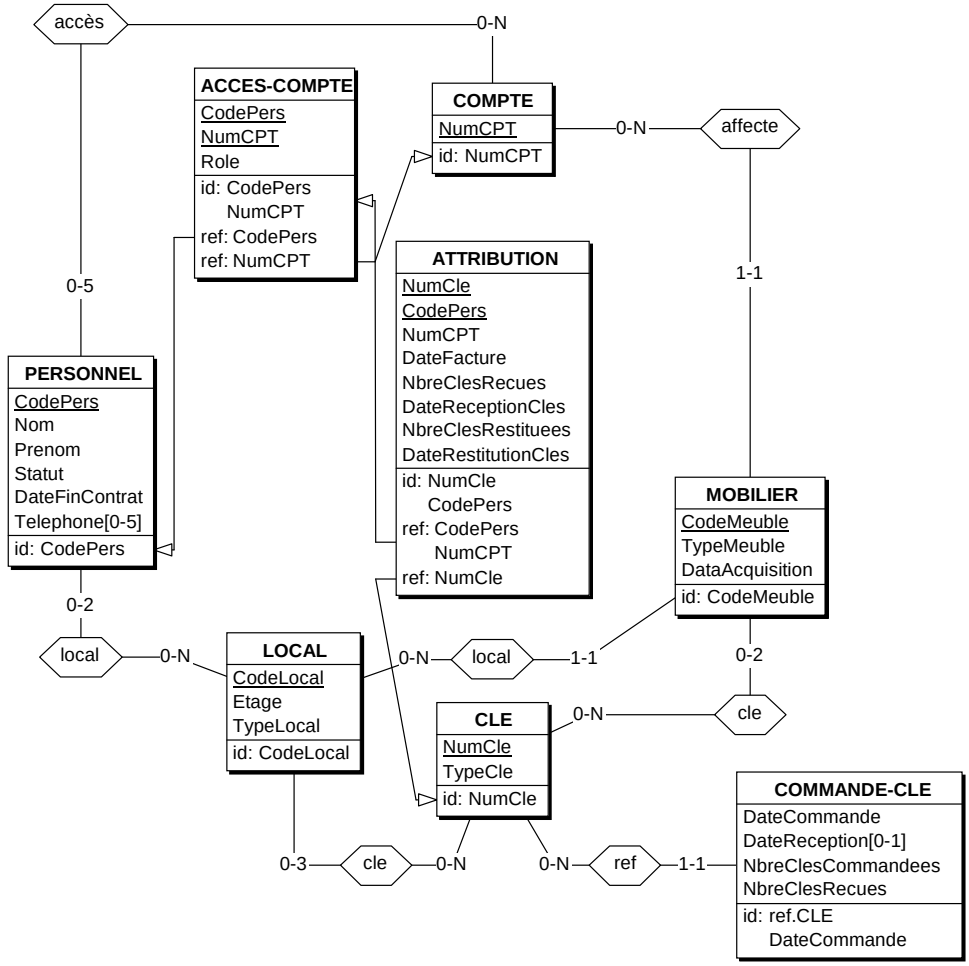


Figure A23.8.3 - Gestion des clés d'une entreprise. Transformations des clés étrangères multivaluées - première phase.

c) Le noyau irréductible

Malheureusement, les autres clés étrangères résistent à la transformation en type d'associations. En effet, la transformation de l'une altère la structure d'autres contraintes. Il reste ainsi un noyau constitué de ATTRIBUTION, ACCES-COMPTE, PERSONNEL, COMPTE et CLE (A23.8.4) qui va nécessiter un traitement spécifique.

Le problème est causé par l'attribut CodePers dans ATTRIBUTION, qui intervient comme l'un des composants d'une clé étrangères et de l'identifiant. On duplique cet attribut de manière à rendre ces deux groupes disjoints. Une contrainte d'égalité garantit que ces deux attributs auront toujours la même valeur (A23.8.5).

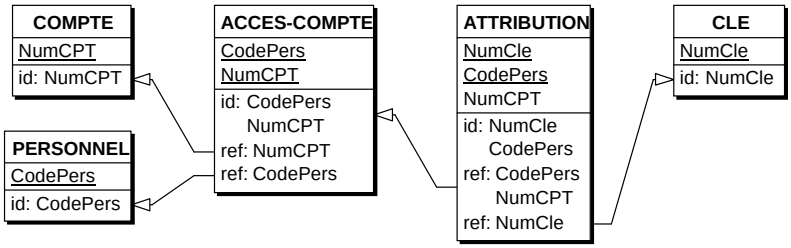


Figure A23.8.4 - Noyau irréductible dans la transformation des clés étrangères

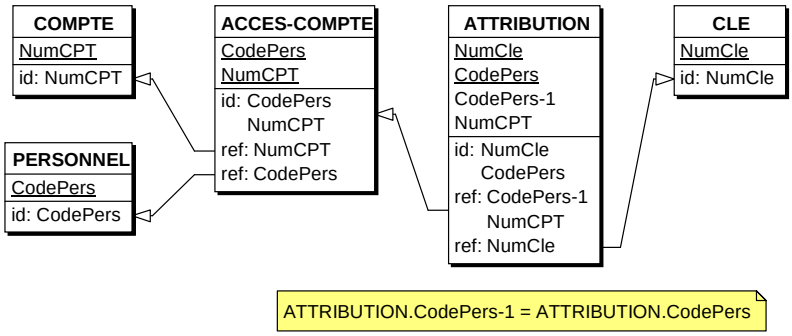


Figure A23.8.5 - Découplage de l'identifiant et d'une clé étrangère de ATTRIBUTION par duplication de l'attribut *CodePers*.

Il est à présent possible de transformer les quatre clés étrangères en types d'associations (A23.8.6). Un exemplaire de l'attribut CodePers subsiste dans ATTRIBUTION. Bien que redondant, il ne peut être supprimé car il intervient dans l'identifiant de ATTRIBUTION. Nous le conservons ainsi que la contrainte de redondance qui en exprime les valeurs.

En examinant de plus près la contrainte de redondance, on observe que ATTRIBUTION.CodePers se comporte comme une clé étrangère vers PERSONNEL. Cette clé ne se substitue cependant pas à la contrainte d'égalité, que nous devons également conserver. En transformant cette clé étrangère en type d'associations, on obtient le schéma de la A23.8.7. La contrainte de redondance est adaptée en conséquence. Elle correspond désormais à une contrainte cyclique. Il serait possible de s'en passer si elle n'intervenait pas dans l'identifiant de ATTRIBUTION.

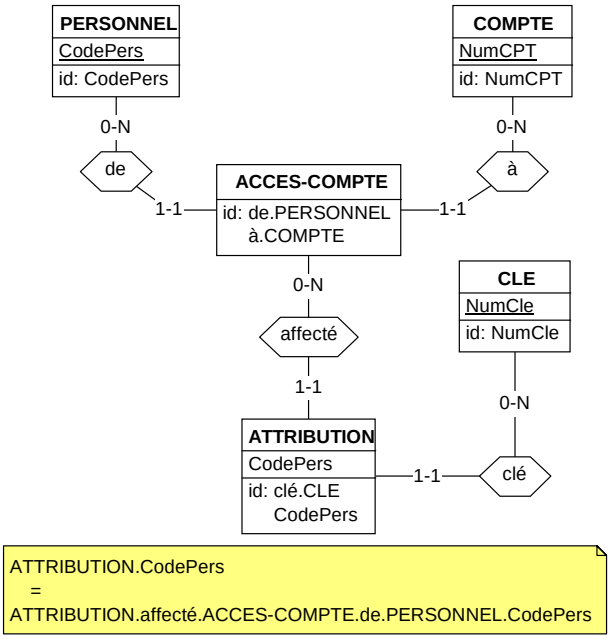


Figure A23.8.6 - Transformation des clés étrangères du noyau irréductible

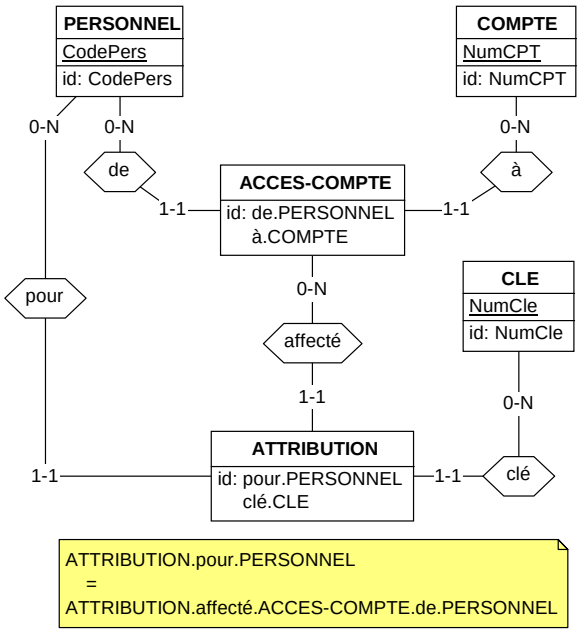


Figure A23.8.7 - Elimination du dernier attribut CodePers de ATTRIBUT

d) Constructions redondantes

Il apparaît que le type d'associations accès entre PERSONNEL et COMPTE est redondant par rapport à la composition des types d'associations PERSONNEL.de.ACCES-COMPTE et ACCES-COMPTE.à.COMPTE. Il faut faire un choix entre les cardinalités [0-N] et [0-5] pour le rôle de.PERSONNEL. Nous choisissons la contrainte la plus générale [0-N], estimant que l'autre résultait des contraintes d'implémentation. On obtient ainsi le schéma conceptuel de la A23.8.8.

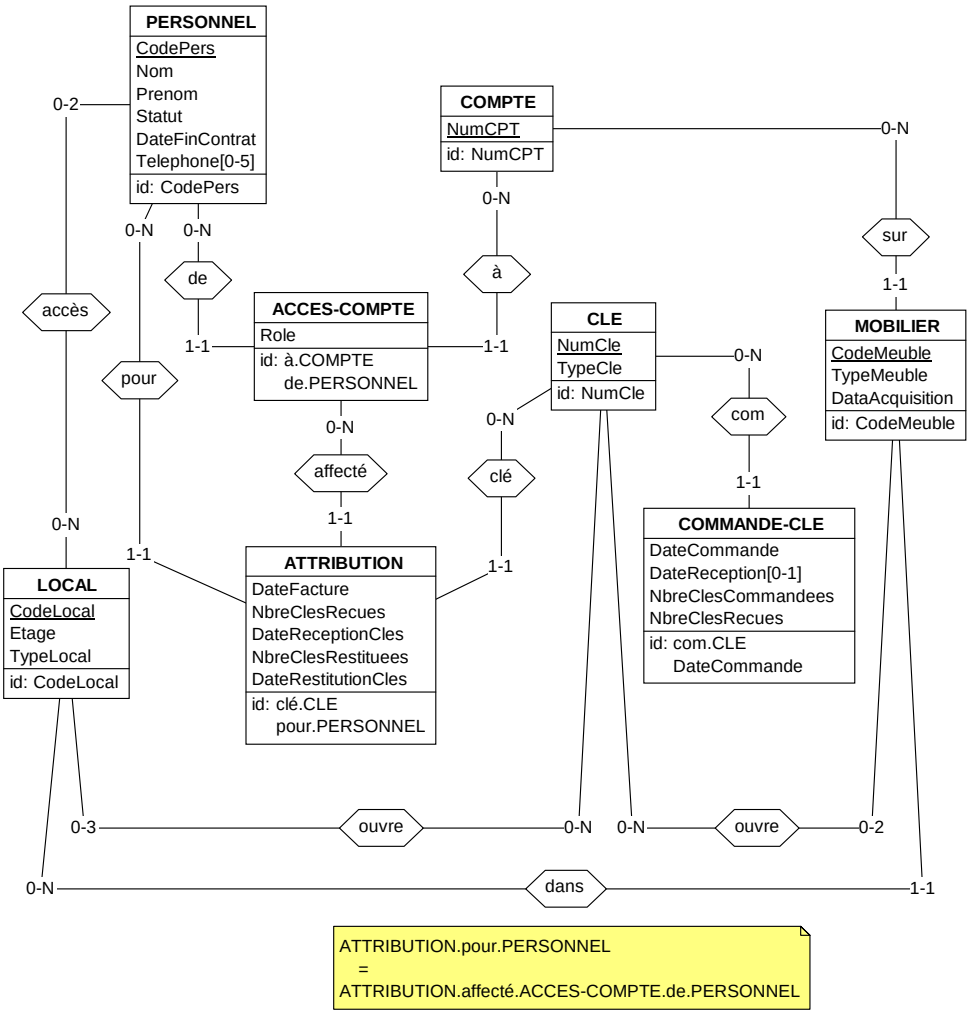


Figure A23.8.8 - Gestion des clés d'une entreprise. Schéma conceptuel final.

A23.9MIGRATION DE FICHIERS

On se propose de convertir les structures de données d'une collection de fichiers standards (COBOL en l'occurrence) en structures de base de données relationnelle, essentiellement par analyse du code source d'une petite application. Nous ignorerons l'exploitation des autres sources d'information qui sont typiquement utilisées lors de la conduite de projets de rétro-ingénierie en vraie grandeur. Nous ignorerons également la migration des données et celle des programmes d'application, processus qui dépasseraient la portée de cet ouvrage.

A23.9.1Démarche de rétro-ingénierie

Les phases principales de la rétro-ingénierie sont rappelées dans le scénario de la A23.9.1.

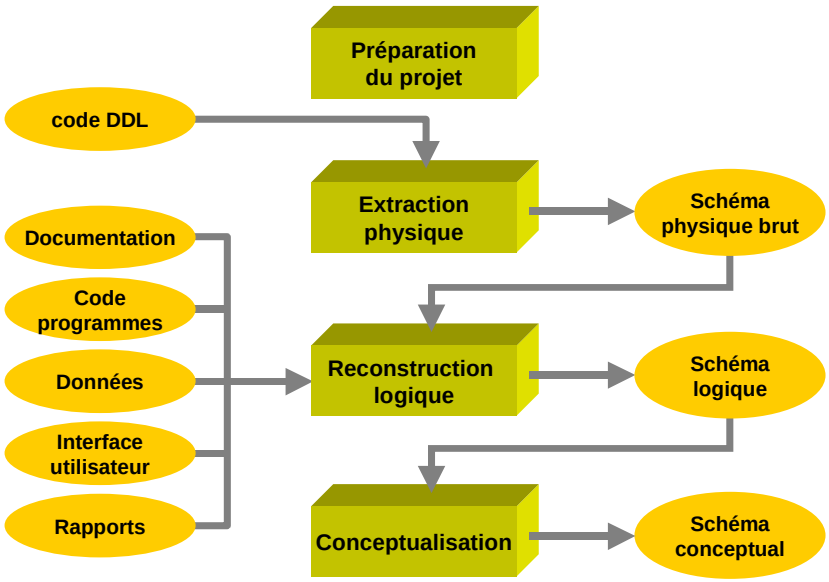


Figure A23.9.1 - Les phases du processus de rétro-ingénierie

a) Préparation du projet

La portée du projet est délimitée. En particulier, on fixe les objectifs, on identifie les sources d'information, les ressources nécessaires, on établit le planning et les techniques de conduite du projet. En particulier, on fixe les critères de validation des produits et de terminaison des processus.

L'un des documents particulièrement utiles dans cette tâche de préparation est l'architecture générale de l'application, qu'il est en général assez aisé de reconstituer.

b) Extraction des structures de données

On produit, à partir du code DDL, un *schéma physique brut* incluant les constructions explicites de la base de données source. Le cas d'une collection de fichiers standard est un peu plus complexe dans la mesure où il n'existe pas de description centralisée des structures complètes des fichiers et des types d'enregistrements. Il faut dans ce cas extraire les déclarations des programmes utilisateurs des fichiers et intégrer ces descriptions.

c) Reconstruction du schéma logique

Le schéma physique brut est enrichi des constructions implicites pour constituer un *schéma logique* complet. Les constructions implicites sont obtenues par analyse de sources complémentaires telles que le code source des programmes, les données, les écrans et les rapports.

On notera que ce schéma logique final est en général plus riche que ce que permettrait le modèle du SGD pris au sens strict. On trouvera par exemple des clés étrangères exprimant les liens de référence implicites dans une collection de fichiers COBOL.

d) Conceptualisation

Cette phase produit un schéma conceptuel qui constitue l'interprétation sémantique la plus probable du schéma logique. Elle comprend trois processus essentiels :

1. la *préparation du schéma*, par laquelle on conditionne le schéma pour en améliorer la lisibilité et l'interprétation;
2. la *conceptualisation du schéma* proprement dite, qui produit par transformations successives un premier schéma conceptuel brut; il comprend deux types de transformations :
   a) *désoptimisation* : on élimine ou on remplace par transformation les structures destinées à optimiser la gestion et l'exploitation des données;
   b) *détraduction* : on remplace les constructions propres au modèle du système de gestion de données par leur interprétation conceptuelle;
3. la *normalisation*, qui transforme le schéma conceptuel brut pour en éliminer les anomalies de représentation et pour en améliorer les qualités de lisibilité et d'expressivité, ou encore pour le rendre conforme à un standard méthodologique déterminé.

A23.9.2Description du projet

Le projet que nous allons réaliser consiste à convertir les structures de données d'un ensemble de fichiers COBOL en une base de données relationnelle équivalente.

On dispose d'une seule source d'information : le texte source d'un programme COBOL qui gère et exploite le contenu de ces fichiers. Ce programme comprend des sections de définition des fichiers et des types d'enregistrements, ce que nous

désignerons par le terme de *code DDL*[14]. Il comprend aussi la définition des variables locales et le code procédural.

Nous construirons un schéma logique aussi complet que possible à partir du code du programme, nous le transformerons en un schéma conceptuel plausible, puis nous traduirons ce dernier en un schéma relationnel exprimé en SQL-DDL.

A23.9.3Préparation du projet

Ce processus consiste essentiellement en l'identification des sources d'information, qui se réduit ici au texte source du programme retenu. Le code complet est repris à la section A23.9.8.

La structure générale du programme est donnée à la A23.9.2. Elle permet d'identifier les portions de programmes qui accèdent aux fichiers.

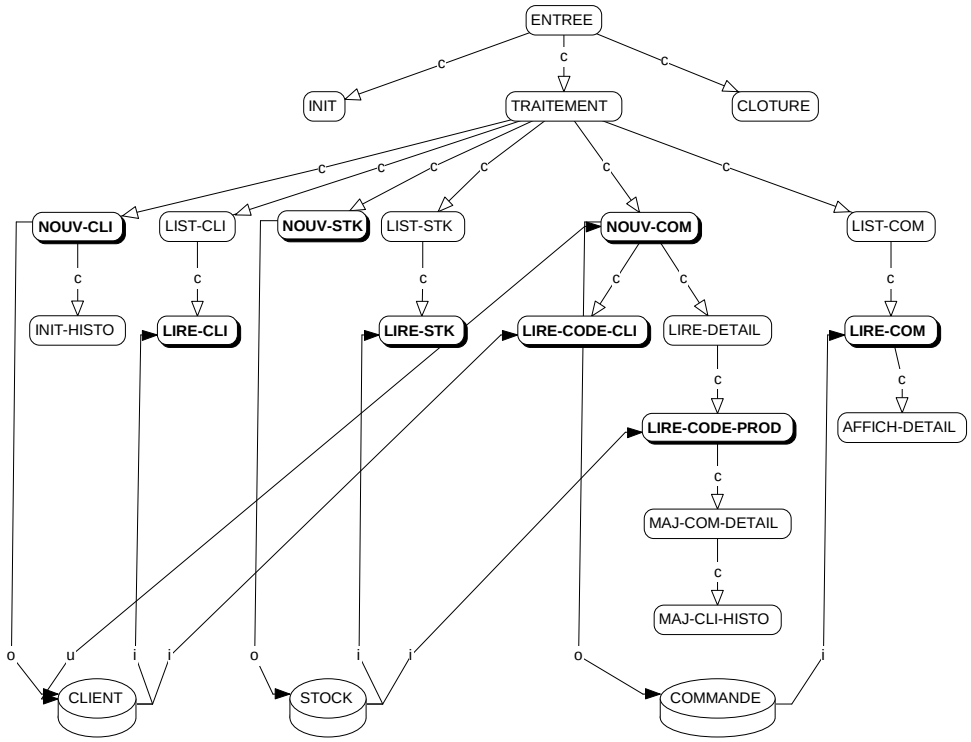


Figure A23.9.2 - Architecture générale du programme. On y représente les liens d'invocation (arcs "c" du graphe des appels, ou *call graph*) entre procédures ainsi que les liens d'interaction avec les fichiers ("i" pour *input*, "o" pour *output* et "u" pour *update*).

14. DDL = Data Definition Language.

A23.9.4Extraction du schéma physique

Nous analyserons les sections DDL du programme pour en extraire le schéma physique brut. Le code DDL d'un programme COBOL comprend deux parties utiles.

- le FILE-CONTROL de l'INPUT-OUTPUT SECTION, qui définit les fichiers utilisés par le programme, leur organisation, leurs index et le caractère identifiant de ces derniers [006-019].
- les FD de la FILE SECTION, qui énumèrent les types d'enregistrements de chaque fichier, leur décomposition en champs et le type de valeurs de chacun d'eux [022-036].

Le résultat de l'interprétation de ce code est présenté à la A23.9.3.

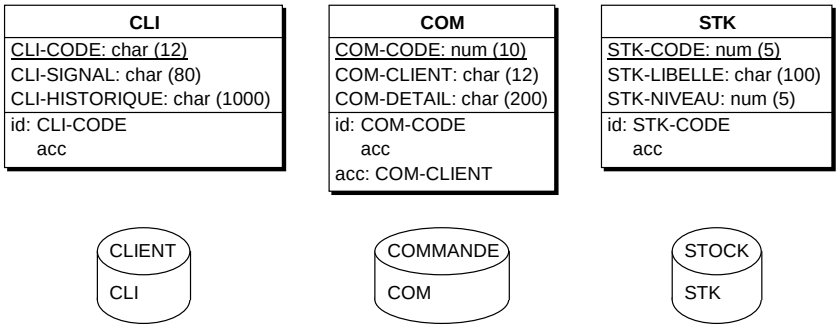


Figure A23.9.3 - Schéma physique brut extrait du code DDL du programme COBOL.

A23.9.5Reconstruction du schéma logique

Nous analyserons ensuite le code procédural pour y découvrir les éventuelles constructions implicites. L'usage de l'organisation séquentielle indexée nous autorise raisonnablement à considérer que tous les identifiants sont déclarés explicitement, de sorte qu'il n'est pas nécessaire de rechercher la présence d'autres identifiants dans cette phase d'affinage.

Nous retiendrons un nombre limité d'objectifs : la *décomposition* précise des champs, les *clés étrangères*, les *identifiants des champs multivalués* et les *cardinalités* précises des champs multivalués. Dans un projet réel, il conviendrait sans doute de rechercher d'autres constructions et contraintes implicites.

a) Décomposition précise des champs

La présence de champs de grande taille suggère que ceux-ci pourraient être dotés d'une structure implicite que nous allons essayer de découvrir. Nous nous intéresserons de plus près aux quatre champs CLI-SIGNAL, CLI-HISTORIQUE, COM-DETAIL et STK-LIBELLE.

L'analyse du diagramme des flux du programme montre que le champ CLI-SIGNAL est en relation avec la variable locale SIGNALETIQUE. On trouve en effet deux instructions, en [105] et [152] (Code 9.1), qui indiquent que, lors de l'exécution du programme, ce champ et cette variable contiennent les mêmes valeurs. On peut donc légitimement penser que la structure de ces valeurs est identique.

```
MOVE SIGNALETIQUE TO CLI-SIGNAL.          [105]
MOVE CLI-SIGNAL TO SIGNALETIQUE.          [152]
```

Code 9.1 - Instructions d'assignation définissant le graphe de flux concernant le champ CLI-SIGNAL.

Nous allons donc assigner au champ CLI-SIGNAL la structure de la variable SIGNALISATION (Code 9.2), ce qu'on consigne dans le schéma physique (A23.9.4).

```
01 SIGNALETIQUE.                          [038]
   02 NOM PIC X(20).                      [039]
   02 ADRESSE PIC X(40).                  [040]
   02 FONCTION PIC X(10).                 [041]
   02 DATE-ENREG PIC X(10).               [042]
```

Code 9.2 - Définition de la variable SIGNALETIQUE.

Selon une technique similaire, nous pouvons affiner la structure des champs CLI-HISTORIQUE et COM-DETAIL. En revanche, l'examen de STK-LIBELLE ne fournit pas d'indice d'une décomposition pertinente, ce qui suggère qu'il s'agit d'un champ atomique. Nous obtenons alors le schéma physique de la A23.9.4.

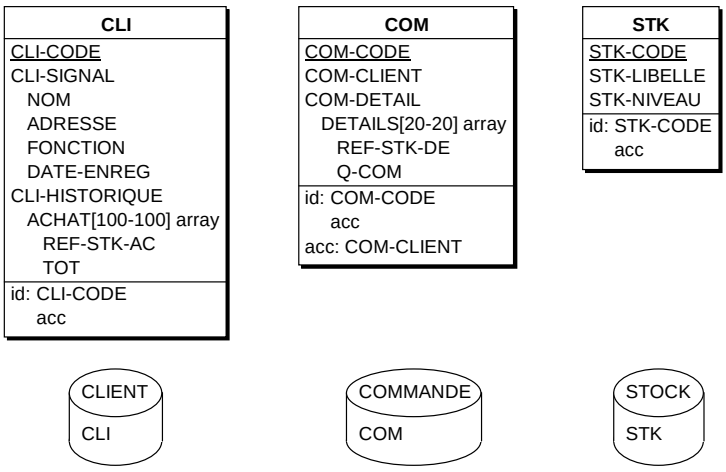


Figure A23.9.4 - Schéma physique : expression des champs composés.

b) Recherche des clés étrangères

Il existe certainement des liens entre ces trois fichiers qui s'expriment sous la forme de clés étrangères implicites. Nous allons rechercher ces liens à l'aide de plusieurs techniques coordonnées.

L'examen du nom des champs peut donner une première indication. En effet le nom d'un champ de référence contient souvent, sous une forme plus ou moins explicite, l'identification du type d'enregistrements cible. Cette propriété nous fait repérer, par exemple, les noms REF-**STK**-AC (contient le nom d'un type d'enregistrement), REF-**STK**-DE (idem) et COM-**CLIENT** (contient le nom d'un fichier). Examinons de plus près le champ COM-**CLIENT**.

1. Comme nous venons de l'observer, son nom fait référence à un fichier contenant des enregistrements CLI.
2. Son type et sa longueur sont ceux de l'identifiant du type d'enregistrements CLI.
3. Il constitue un index, ce qui est fréquent pour les identifiants et les clés étrangères.
4. Il apparaît dans le diagramme de flux du programme en association avec la *record key* (index identifiant) du fichier CLIENT (Code 9.1).
5. L'analyse de l'instruction WRITE COM en [160] montre qu'un enregistrement COM n'est écrit dans le fichier COMMANDE qu'après qu'on ait vérifié que la valeur de COM-CLIENT correspond à un enregistrement existant dans le fichier CLIENT (Code 9.2).

```
MOVE CLI-CODE TO COM-CLIENT.                    [154]
```

Code 9.1 - Instruction d'assignation définissant le graphe de flux concernant le champ COM-CLIENT.

```
NOUV-COM.                                       [146]
   MOVE 1 TO END-FILE.                          [150]
   PERFORM LIRE-CODE-CLI UNTIL END-FILE = 0.    [151]
   MOVE CLI-CODE TO COM-CLIENT.                 [154]
   WRITE COM                                    [160]
     INVALID KEY DISPLAY "ERREUR".              [161]

LIRE-CODE-CLI.                                  [165]
   ACCEPT CLI-CODE.                             [167]
   MOVE 0 TO END-FILE.                          [168]
   READ CLIENT INVALID KEY                      [169]
      DISPLAY "CLIENT INEXITANT"                [170]
      MOVE 1 TO END-FILE                        [171]
   END-READ.                                    [172]
```

Code 9.2 - Fragment de programme (*program slice*) relatif à l'objet COM.COM-CLIENT au point [160]. Ce code illustre une validation suggérant une contrainte référentielle.

Sur base de ces cinq indices, nous pouvons en confiance conclure que COM-CLIENT doit être une clé étrangère vers CLI. Par des raisonnements similaires, nous ferons de COM-DETAIL.DETAILS.REF-STK-DE et CLI-HISTORIQUE.ACHAT.REF-STK-AC des clés étrangères vers STK, ce qui conduit au schéma physique augmenté de la A23.9.5.

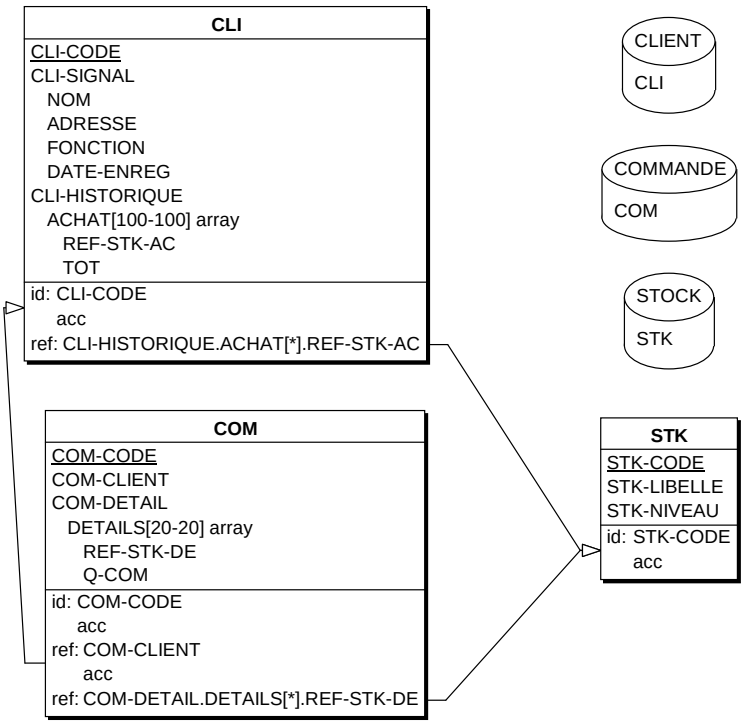


Figure A23.9.5 - Schéma physique : expression des clés étrangères.

c) Recherche des identifiants des champs multivalués

Le schéma physique contient deux champs multivalués composés, DETAILS et ACHAT. Il est fréquent que les valeurs de tels champs soient soumises à une contrainte d'unicité portant sur un de leurs composants.

En examinant la manière dont le programme lit et gère les valeurs de ces champs, nous pouvons tirer des renseignements utiles concernant la présence éventuelle d'identifiants locaux. Considérons par exemple le champ multivalué DETAILS et calculons le fragment de programme pour ce champ au point WRITE COM [160] (Code 9.1), qui devrait nous apprendre comment ce champ est garni préalablement à l'écriture de l'enregistrement dans le fichier.

Ce fragment montre clairement ([190] à [204]) qu'il ne peut exister deux éléments DETAILS de même valeur de REF-STK-DE. On en conclut donc que ce champ est un identifiant de COM-DETAIL.DETAILS.

On montrerait de la même manière que REF-STK-AC est un identifiant du champ multivalué CLI-HISTORIQUE.ACHAT. Ces découvertes sont incorporées au schéma physique courant (A23.9.6).

```
ENTREE.                                          [059]
   PERFORM TRAITEMENT UNTIL CHOIX = 0.           [061]
TRAITEMENT.                                      [068]
   IF CHOIX = 3                                  [081]
      PERFORM NOUV-COM.                          [082]
NOUV-COM.                                        [146]
   SET IND-DET TO 1.                             [156]
   MOVE 1 TO END-FILE.                           [157]
   PERFORM LIRE-DETAIL
           UNTIL END-FILE = 0 OR IND-DET = 21.   [158]
   MOVE LIST-DETAIL TO COM-DETAIL.               [159]
   WRITE COM                                     [160]
LIRE-DETAIL.                                     [173]
   ACCEPT CODE-PROD.                             [175]
   IF CODE-PROD = "0"                            [176]
      MOVE 0 TO END-FILE                         [177]
      MOVE 0 TO REF-STK-DE(IND-DET)              [178]
   ELSE                                          [179]
      PERFORM LIRE-CODE-PROD.                    [180]
LIRE-CODE-PROD.                                  [181]
   MOVE 1 TO EXIST-PROD.                         [182]
   MOVE CODE-PROD TO STK-CODE.                   [183]
   READ STOCK INVALID KEY                        [184]
      MOVE 0 TO EXIST-PROD.                      [185]
   IF EXIST-PROD = 0                             [186]
      DISPLAY "PRODUIT INEXISTANT"               [187]
   ELSE                                          [188]
      PERFORM MAJ-COM-DETAIL.                    [189]
MAJ-COM-DETAIL.                                  [190]
   MOVE 1 TO NEXT-DET.                           [191]
   ACCEPT Q-COM(IND-DET).                        [193]
   PERFORM UNTIL                                 [194]
            REF-STK-DE(NEXT-DET) = CODE-PROD     [195]
         OR IND-DET = NEXT-DET                   [196]
      ADD 1 TO NEXT-DET                          [197]
   END-PERFORM.                                  [198]
   IF IND-DET = NEXT-DET                         [199]
      MOVE CODE-PROD TO REF-STK-DE(IND-DET)      [200]
      SET IND-DET UP BY 1                        [202]
   ELSE                                          [203]
      DISPLAY "ERREUR : PRODUIT DEJA COMMANDE".  [204]
```

Code 9.1 - Fragment de programme (*program slice*) relatif à l'objet COM.COM-DETAIL.DETAILS au point [160]. Ce code illustre une validation contrôlant l'unicité des valeurs de REF-STK-DE.

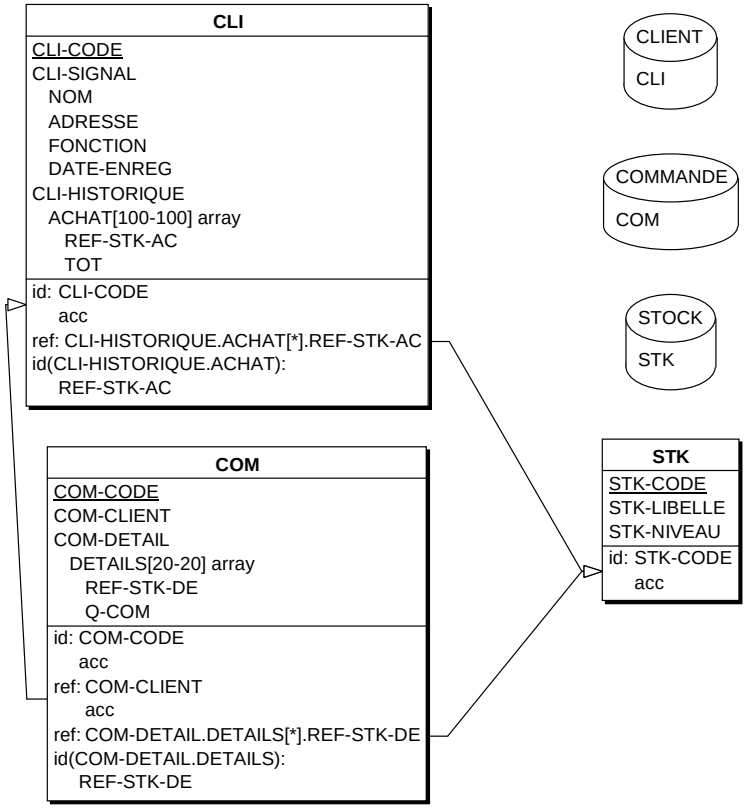


Figure A23.9.6 - Schéma physique : expression des identifiants des champs multivalués.

d) Détermination des cardinalités exactes des champs multivalués

Les champs multivalués ont été interprétés comme des tableaux[15] contenant un nombre fixe de valeurs, respectivement [20-20] et [100-100]. Ces bornes sont-elles effectivement d'application ?

Le fragment de programme Code 9.1 répond à cette question pour le champ DETAILS. On observe en effet que le tableau peut contenir jusqu'à 20 valeurs, mais que le programme accepte que le tableau soit vide. On en conclut que la cardinalité exacte de DETAILS est [0-20]. On montrerait de manière semblable que celle de ACHAT est [0-100]. Ces précisions sont ajoutées au schéma physique de la A23.9.7, qui devient ainsi le schéma physique complet, malgré les réserves qu'on pourrait formuler sur la notion de *complétude*.

15. En réalité, il s'agit de *unique arrays* (`u-array`), étant donné que chacun est doté d'un identifiant. Nous ignorerons cette spécification qui n'apporte rien du point de vue sémantique.

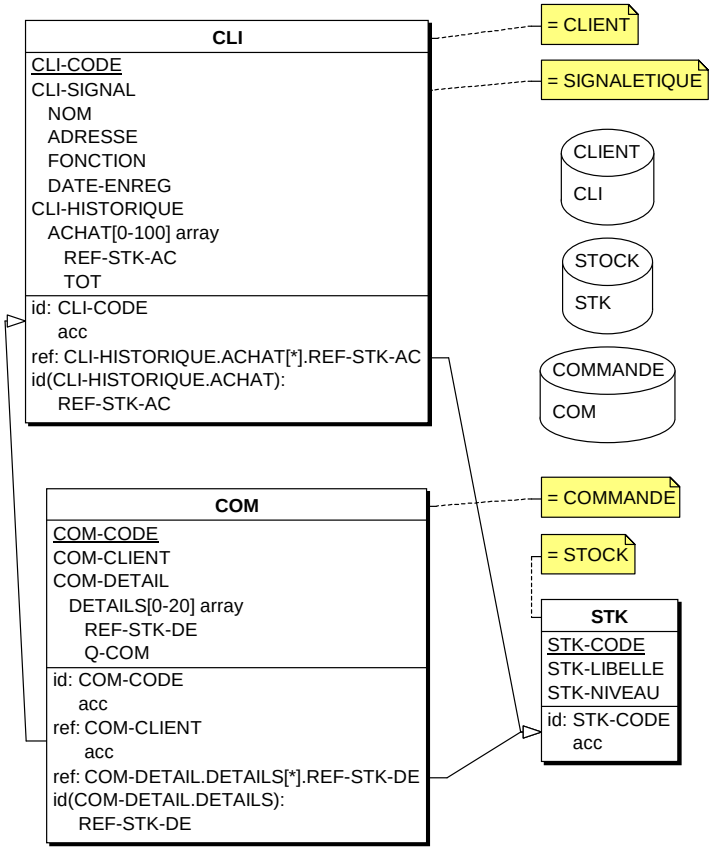


Figure A23.9.7 - Schéma physique affiné et schéma logique COBOL - Expression des cardinalités exactes et noms significatifs.

e) Recherche de noms significatifs

L'analyse des programmes et celle du schéma nous donne l'occasion d'affecter aux objets de celui-ci des noms plus évocateurs que leur domination d'origine. C'est ainsi que :

1. le nom de chaque fichier est plus significatif que celui du type des enregistrements qu'il contient : CLIENT pour CLI, STOCK pour STK et COMMANDE pour COM;
2. une variable qui reçoit, ou qui émet des valeurs d'un champ peut avoir un nom plus clair que celui du champ en question; tel est le cas de la variable SIGNALETIQUE, qui est en relation avec le champ CLI-SIGNAL.

Nous éviterons à ce stade d'effectuer la substitution, qui rendrait inopérant le schéma logique. Celui-ci peut en effet être utilisé pour maintenir les programmes et pour en développer d'autres. Il est donc important de conserver les noms d'origine. Nous

nous contenterons pour l'instant d'*annoter* le schéma, laissant à la phase de conceptualisation le soin d'effectuer la modification des noms.

f) Extraction du schéma logique complet

En toute généralité, le schéma logique se déduit du schéma physique par l'élimination des constructions physiques non pertinentes au niveau logique. Cette notion dépend cependant du modèle selon lequel le schéma physique est exprimé. Dans notre cas le schéma relève du modèle COBOL, qui inclut les concepts d'index et de fichier au niveau logique[16]. Dans le cas d'une base de données SQL, il faudrait supprimer ces constructions pour obtenir le schéma logique.

Le schéma physique de la A23.9.7 est donc aussi le schéma logique COBOL.

A23.9.6Conceptualisation des structures de données

Le schéma logique de la A23.9.7 va être retravaillé pour produire un schéma conceptuel raisonnable.

a) Préparation du schéma logique

Le schéma logique comporte des traces des opérations d'extraction et d'affinage qui sont sans intérêt pour le recouvrement des structures conceptuelles et qui risquent d'obscurcir le travail ultérieur. Nous les éliminerons de manière à disposer d'un schéma clair et expressif[17]. Nous appliquerons trois techniques.

• Traitement des noms

Préfixes non significatifs. Nous observons que les noms des champs de niveau supérieur sont systématiquement préfixés par celui de leur type d'enregistrement (**CLI**-SIGNAL). Il s'agit d'une pratique courante de formation de noms uniques en COBOL qui évite de devoir qualifier le nom des composants lorsqu'il apparaît dans un programme[18]. Ces préfixes n'apportant aucune information, on propose de les éliminer.

Noms significatifs. On effectue la substitutions des noms significatifs mis en évidence sous la forme d'annotations.

16. Rappelons qu'on qualifie de logique un schéma qui contient les éléments nécessaires et suffisants pour que le programmeur puisse rédiger les programmes d'application. En COBOL, la connaissance des fichiers et des index est nécessaire, tandis qu'en SQL elle ne l'est pas. Il est donc normal que les index et fichiers apparaissent dans un schéma logique COBOL.
17. Ce processus n'est pas étranger à celui de normalisation.
18. On peut écrire CLI-SIGNAL au lieu de CLI-SIGNAL of CLI si ce nom est unique, ce qui rend plus concises les instructions d'un langage généralement verbeux.

• Transformation des structures anormales

La technique d'affinage de la structure des champs longs a créé deux champs composés qui ne possèdent qu'un seul composant, soient CLI-HISTORIQUE et COM-DETAIL . On propose de remplacer ces champs composés par cet unique composant.

• Suppression des constructions physiques

Les clés d'accès (index) et les fichiers n'ont plus d'utilité à ce stade, et peuvent être supprimés du schéma logique.

Nous obtenons de la sorte un schéma logique préparé qui peut être conceptualisé (A23.9.8).

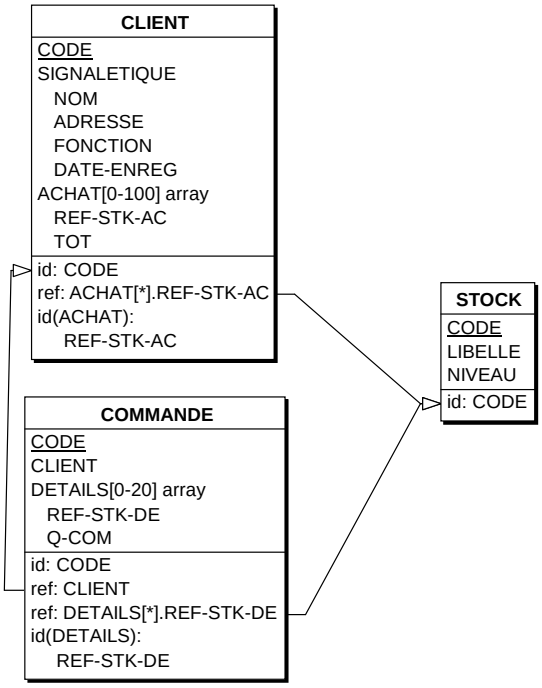


Figure A23.9.8 - Schéma logique nettoyé.

b) Désoptimisation et détraduction du schéma logique

Les processus de *désoptimisation* et de *détraduction* sont en général intimement liés, il est inutile de chercher à les distinguer sous la forme de processus séquentiels indépendants.

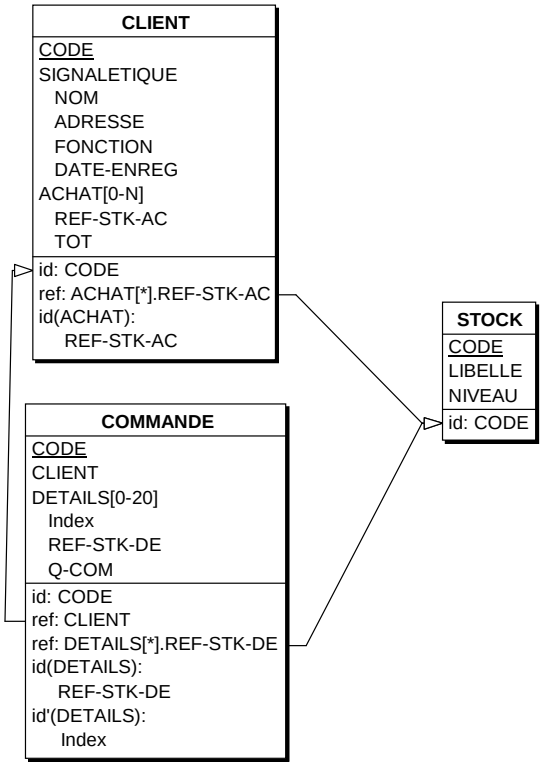


Figure A23.9.9 - Schéma partiellement conceptualisé : traitement des tableaux et des cardinalités.

Les tableaux. Le langage COBOL, à l'instar de la plupart des langages procéduraux, ne dispose que de la structure de *tableau* pour représenter explicitement des attributs multivalués. Il importe de préciser l'interprétation réelle des deux tableaux détectés dans les types d'enregistrements CLIENT et COMMANDE.

- ACHAT[0-100] : on admet que la structure d'ordre et le concept de *cellule vide*[19] sont sans signification; il s'agit donc d'un simple ensemble;
- DETAILS[0-20] : la notion d'ordre semble importante mais pas celle de *cellule vide*, ce qui suggère de remplacer la structure de tableau par celle de liste. Cette dernière est exprimée sous une forme ensembliste par l'introduction d'un composant Index (A23.9.9).

Les cardinalités. La cardinalité maximum de 100 de ACHAT n'a pas d'autre justification qu'une limite d'occupation d'espace dans la structure des

19. On rappelle qu'un tableau est une collection de cellules indexées pouvant contenir chacune une valeur. Le tableau a donc pour caractéristiques, par rapport à l'ensemble de valeurs : pas d'unicité, ordre et possibilité de *trous* dans la séquence (cellules vides).

enregistrements. On peut donc la considérer comme résultant d'une décision d'optimisation. On la remplace par N. En revanche, il apparaît que la limite de 20 de DETAILS a une justification organisationnelle (on évite les commandes trop longues, considérées comme anormales). On propose donc de la conserver (A23.9.9).

Les attributs multivalués complexes. Les deux attributs complexes DETAIL et ACHAT correspondent à une implémentation typique de types d'entités dépendants. Cette structure permet de diminuer le nombre d'accès lorsqu'on accède à une commande et à tous ses détails. Il s'agit de structures d'optimisation qu'on traite par transformation en types d'entités DETAILS et ACHAT (A23.9.10).

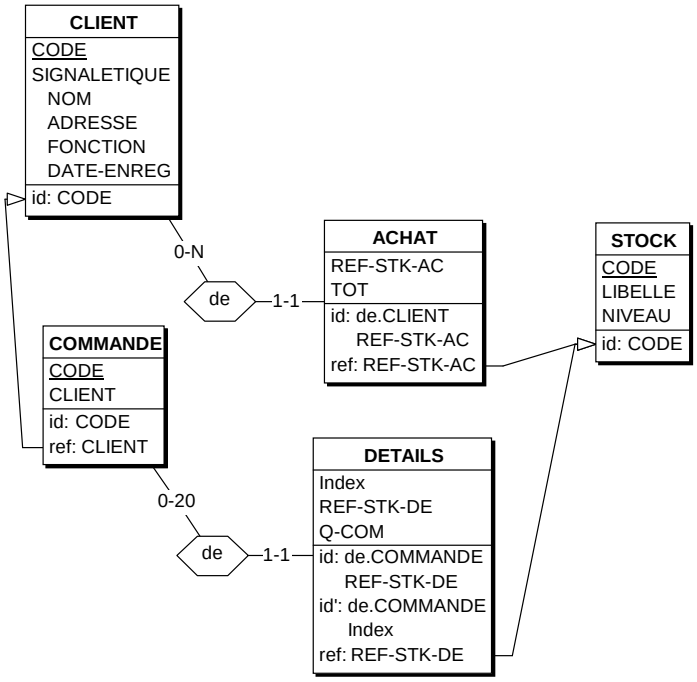


Figure A23.9.10 - Schéma partiellement conceptualisé : les attributs multivalués complexes sont transformés en types d'entités dépendants.

Les clés étrangères. Les clés étrangères sont remplacées par le type d'associations dont elle est l'implémentation. Ces clés étant mono-valuées et non identifiantes, les types d'associations résultants sont *un-à-plusieurs* (A23.9.11).

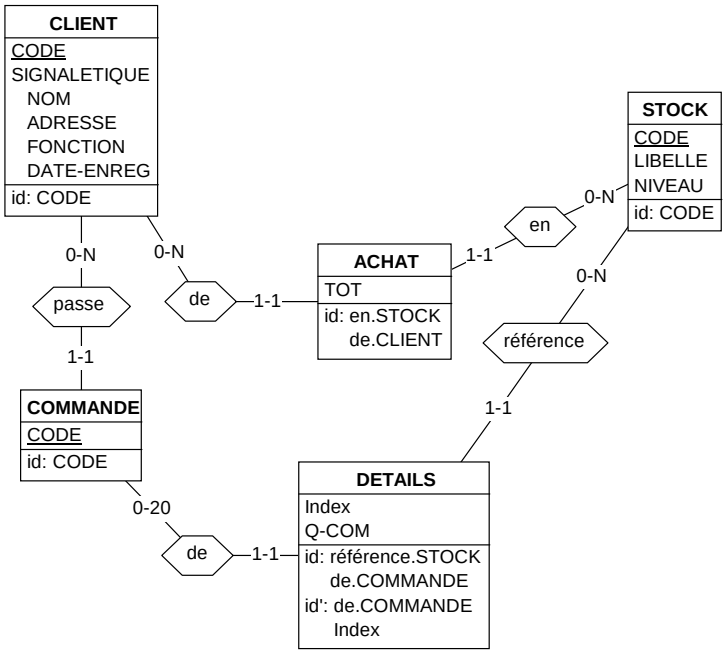


Figure A23.9.11 - Schéma conceptuel brut. Les clés étrangères sont transformées en types d'associations *un-à-plusieurs*.

c) Normalisation du schéma conceptuel

Nous tenterons à présent de rendre ce schéma plus lisible, de lui donner des qualités de concision, de minimalité, et d'expressivité, et d'éliminer les éventuelles constructions maladroites. Nous appliquerons quelques transformations qui nous semblent pertinentes.

Types d'entités associations. Les entités ACHAT et DETAILS pourraient être considérées comme jouant un rôle d'associations entre les entités CLIENT et STOCK d'une part et entre COMMANDE et STOCK d'autre part. On peut donc envisager de les réduire à de simples types d'associations. Nous le ferons pour ACHAT, mais pas pour DETAILS, qui nous semble plus complexe, et mieux exprimé sous sa forme actuelle.

Traitement des noms. Certains noms mériteraient d'être rendus plus expressifs. On pense en particulier à l'attribut Index, de nature technique, qu'on renomme en Num-Détail.

De même, l'attribut TOT peut être complété en TOTAL.

Enfin, les différents identifiants dénommés CODE pourraient recevoir des noms plus discriminants, tels que CODE-CLIENT, CODE-COMMANDE et CODE-STOCK.

Le terme DETAILS au pluriel est en contradiction avec l'usage généralisé du singulier. On le remplace par DETAIL.

Structures maladroites. L'attribut monovalué SIGNALETIQUE regroupe les informations signalétiques des clients. Ces informations étant, en dehors de l'identifiant, les seules présentes, ce regroupement ne se justifie pas. On désagrège cet attribut.

L'identifiant primaire de DETAIL est porteur de plus d'information, et donc moins stable, que l'identifiant secondaire. On suggère d'échanger leur statut.

Standards méthodologique et documentaire. Nous adopterons nos conventions habituelles de formation des noms, qui veulent qu'un nom d'attribut apparaisse en lettres capitalisé. Le schéma final normalisé est celui de la A23.9.12.

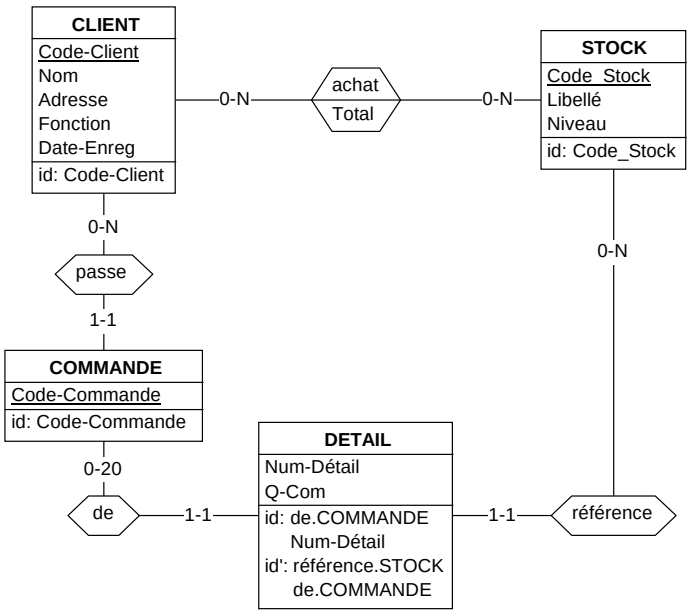


Figure A23.9.12 - Schéma conceptuel normalisé.

A23.9.7Production du schéma relationnel

Nous développerons une base de données de structure simple, dérivant le plus directement possible du schéma conceptuel. Nous veillerons cependant à produire un schéma physique qui respecte toutes les spécifications exprimées dans le schéma de la A23.9.12.

a) Production du schéma logique relationnel

Le schéma conceptuel ne pose aucune problèmes particuliers, et se transforme aisément dans le schéma logique de la A23.9.13. La contrainte de cardinalité [0-20] est représentée par une annotation.

Les noms sont convertis de la manière suivante :

- les accents sont enlevés,
- les minuscules sont converties en majuscule,
- les espaces et tirets sont remplacés par le signe '_',
- les noms correspondant à un mot réservé sont modifiés.

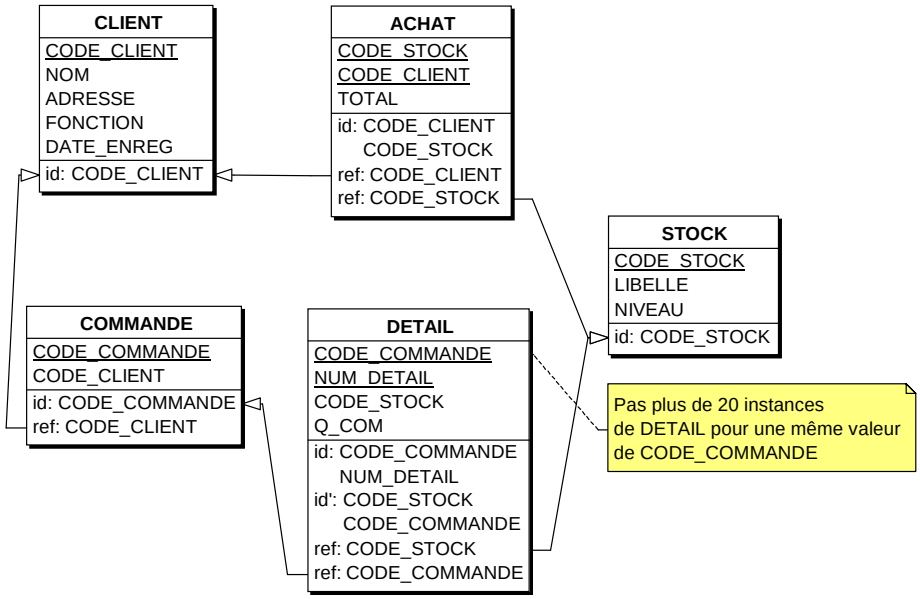


Figure A23.9.13 - Schéma logique relationnel.

b) Production du schéma physique relationnel

Cette phase se limitera à la définition des index, des espaces de stockage et des *triggers*. Les mécanismes d'accès doivent être au moins équivalents à ceux des struc-

tures des fichiers sources. Cependant, on tiendra également compte de besoins nouveaux. Nous veillerons à ne pas déclarer d'index inutiles.

• Les index

On observera d'abord que les fichiers de données que nous avons analysés offrent des mécanismes d'accès qui correspondraient, dans le schéma relationnel, aux index suivants :

- CLIENT.CODE_CLIENT, correspondant à la *record key* du fichier CLIENT,
- COMMANDE.CODE_COMMANDE, correspondant à la *record key* du fichier COMMANDE,
- COMMANDE.CODE_CLIENT, correspondant à l'*alternate record key* du fichier COMMANDE,
- STOCK.CODE_STOCK, correspondant à la *record key* du fichier STOCK,
- DETAIL.CODE_COMMANDE, correspondant à l'inclusion de DETAILS dans COM,
- ACHAT.CODE_CLIENT, correspondant à l'inclusion de ACHAT dans CLI.

Ces index sont représentés à la A23.9.14.

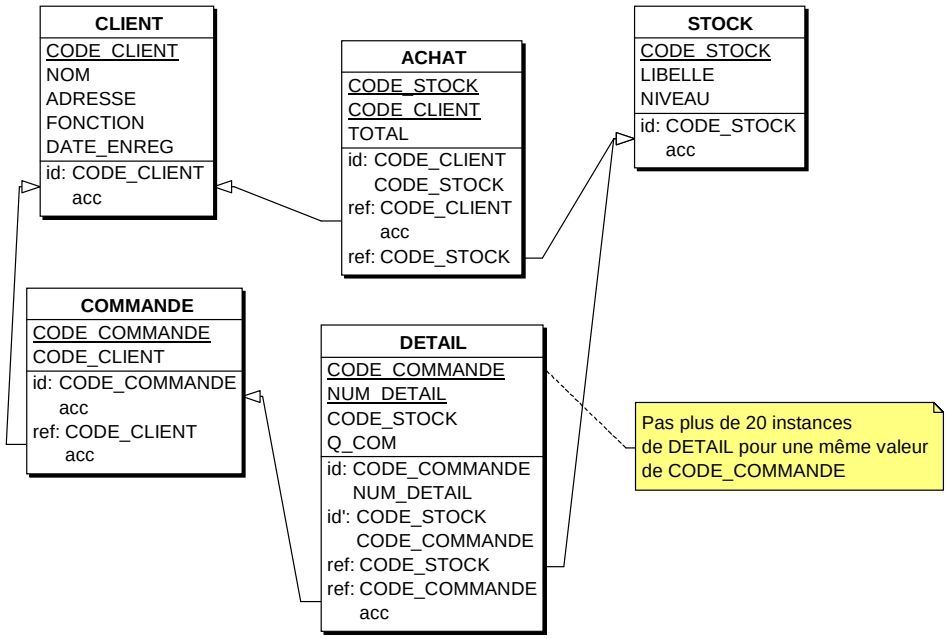


Figure A23.9.14 - Schéma physique - Première version (accès identiques à ceux des fichiers COBOL).

Pour compléter ce schéma, nous ferons l'hypothèse que les achats seront progressivement utilisés pour analyser les produits en stock, de sorte qu'un index sur ACHAT.CODE_STOCK devient également nécessaire. Nous admettrons également que l'accès à DETAIL à partir de STOCK prendra de plus en plus d'importance, ce qui se traduit par un index sur DETAIL.CODE_STOCK.

Outre ces index, nous définirons ceux qui sont nécessaires à la validation des identifiants. Il faudrait en principe définir trois index supplémentaires pour les identifiants d'ACHAT et de DETAIL. C'est ce que nous ferons pour les deux identifiants primaires. Quant à l'identifiant secondaire de DETAIL, nous vérifierons la propriété d'unicité qu'il induit via un système de *triggers*, plus léger qu'un index.

Il reste à éliminer les index préfixes, c'est-à-dire ceux dont les composants apparaissent en premier dans d'autres index. Ces index sont en effet inutiles, les index qui les englobent pouvant être utilisés lors de l'exécution des requêtes[20]. Ceci concerne ACHAT.CODE_CLIENT et DETAIL.CODE_COMMANDE que nous retirons du schéma.

Le jeu d'index proposé est illustré à la A23.9.15.

• Les espaces de stockage

Nous définirons trois espaces de stockage[21] : SP_CLIENT réservé à la table CLIENT, SP_STOCK contenant les tables STOCK et ACHAT, SP_COMMANDE rassemblant les tables COMMANDE et DETAIL. Ces décisions obéissent aux raisonnements suivants :

- la table CLIENT est isolée des autres en raison d'un profil d'exploitation spécifique (backup, profil de mise à jour, type de consultation, etc.);
- les tables COMMANDE et DETAIL sont naturellement fortement couplées, et peuvent faire l'objet d'une politique de stockage entrelacé (*clustering*) qui favorisera leur jointure;
- la table ACHAT a été placée dans l'espace de la table PRODUIT pour favoriser l'analyse des achats.

• Les contraintes d'intégrité additionnelles

Le contrôle de l'identifiant secondaire de DETAIL et la contrainte de cardinalité [0-20] seront pris en charge par des *triggers*.

Identifiant DETAIL.{CODE_STOCK,CODE_COMMANDE}. Deux événements sont susceptibles de perturber cette contrainte d'unicité, l'insertion d'une ligne de DETAIL et la modification des colonnes CODE_STOCK ou CODE _COMMANDE.

20. Pour autant que l'index englobant soit implémenté par une technique de *B-tree*, qui définit un ordre trié des valeurs de l'index, et non par *hashing*.
21. Terme générique désignant les conteneurs tels que les *table-spaces*, *db-spaces* et autres *spaces*.

Cardinalité [0-20] de la clé étrangère DETAIL.CODE_COMMANDE. Les deux événements concernés sont l'insertion d'une ligne de DETAIL et la modification de la colonne CODE_COMMANDE.

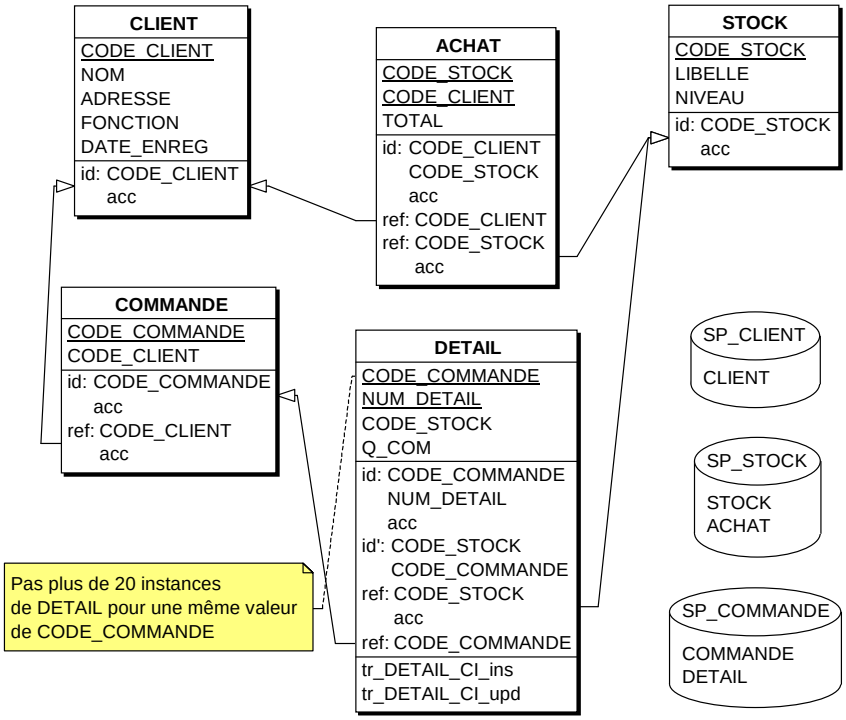


Figure A23.9.15 - Schéma physique relationnel final.

Ces contraintes peuvent être prises en charge par deux jeux de *triggers* indépendants, chacun dédié à une contrainte, ou par un seul jeu assurant le respect des deux contraintes simultanément. Nous choisirons la seconde solution, qui minimise le nombre de *triggers*. Nous définissons donc les deux *triggers* tr_DETAIL_CI_ins et tr_DETAIL_CI_upd attachés à la table DETAIL et déclenchés respectivement par les événements insert et update.

Le schéma final est présenté à la A23.9.15.

c) Génération du code SQL

Nous choisirons un type de codage simple correspondant approximativement au standard SQL2.

• Définition des identifiants

Les identifiants primaires sont déclarés primary keys. L'identifiant secondaire de DETAIL sera géré par les *triggers* de contrôle des contraintes d'intégrité.

• Définition des clés étrangères

Afin d'éviter toute référence en avant, nous déclarerons les clés étrangères par des instructions de modification des tables sources (alter table add constraint).

• Définition des autres contraintes

Comme décidé dans l'élaboration du schéma physique, nous coderons deux triggers gérant les deux contraintes d'intégrité additionnelles affectant la table DETAIL.

• Définition des index

En ce qui concerne les identifiants primaires, la plupart des SGBD exigent qu'ils soient supportés par des index, ce qui a été acté dans le schéma physique. Cependant, ils se différencient par le mode de déclaration. Pour certains, un index est automatiquement construit pour chaque identifiant primaire. Pour les autres, il est nécessaire de déclarer ces index explicitement. Nous ferons l'hypothèse que les index primaires sont construits automatiquement par le SGBD, comme c'est le cas pour Oracle par exemple.

Nous ne coderons donc pas les index identifiants assignés aux *primary keys*.

• Le code SQL-DDL

Le texte intégral du code est reporté à la la section A23.9.9.

A23.9.8*Complément* - Le code source du programme COBOL

```
IDENTIFICATION DIVISION.                                        001
PROGRAM-ID. CLIENT-COMMANDE.                                    002
ENVIRONMENT DIVISION.                                           003
INPUT-OUTPUT SECTION.                                           004

FILE-CONTROL.                                                   005
    SELECT CLIENT ASSIGN TO "CLIENT.DAT"                        006
        ORGANIZATION IS INDEXED                                 007
        ACCESS MODE IS DYNAMIC                                  008
        RECORD KEY IS CLI-CODE.                                 009

    SELECT COMMANDE ASSIGN TO "COMMANDE.DAT"                    010
        ORGANIZATION IS INDEXED                                 011
        ACCESS MODE IS DYNAMIC                                  012
        RECORD KEY IS COM-CODE                                  013
        ALTERNATE RECORD KEY IS COM-CLIENT                      014
        WITH DUPLICATES.                                        015

    SELECT STOCK ASSIGN TO "STOCK.DAT"                          016
        ORGANIZATION IS INDEXED                                 017
        ACCESS MODE IS DYNAMIC                                  018
        RECORD KEY IS STK-CODE.                                 019

DATA DIVISION.                                                  020
FILE SECTION.                                                   021
FD CLIENT.                                                      022
01 CLI.                                                         023
   02 CLI-CODE PIC X(12).                                       024
   02 CLI-SIGNAL PIC X(80).                                     025
   02 CLI-HISTORIQUE PIC X(1000).                               026

FD COMMANDE.                                                    027
01 COM.                                                         028
   02 COM-CODE PIC 9(10).                                       029
   02 COM-CLIENT PIC X(12).                                     030
   02 COM-DETAIL PIC X(200).                                    031

FD STOCK.                                                       032
01 STK.                                                         033
   02 STK-CODE PIC 9(5).                                        034
   02 STK-LIBELLE PIC X(100).                                   035
   02 STK-NIVEAU PIC 9(5).                                      036
```

```
WORKING-STORAGE SECTION.                                    037
01 SIGNALETIQUE.                                            038
   02 NOM PIC X(20).                                        039
   02 ADRESSE PIC X(40).                                    040
   02 FONCTION PIC X(10).                                   041
   02 DATE-ENREG PIC X(10).                                 042

01 LIST-ACHAT.                                              043
   02 ACHAT OCCURS 100 TIMES INDEXED BY IND.                 044
      03 REF-STK-AC PIC 9(5).                                045
      03 TOT PIC 9(5).                                       046

01 LIST-DETAIL.                                             047
   02 DETAILS OCCURS 20 TIMES INDEXED BY IND-DET.            048
      03 REF-STK-DE PIC 9(5).                                049
      03 Q-COM PIC 9(5).                                     050

01 CHOIX PIC X.                                             051
01 END-FILE PIC 9.                                          052
01 END-DETAIL PIC 9.                                        053
01 EXIST-PROD PIC 9.                                        054
01 CODE-PROD PIC 9(5).                                      055
01 QTE PIC 9(5) COMP.                                       056
01 NEXT-DET PIC 99.                                         057

PROCEDURE DIVISION.                                         058
ENTREE.                                                     059
   PERFORM INIT.                                            060
   PERFORM TRAITEMENT UNTIL CHOIX = 0.                      061
   PERFORM CLOTURE.                                         062
   STOP RUN.                                                063

INIT.                                                       064
   OPEN I-O CLIENT.                                         065
   OPEN I-O COMMANDE.                                       066
   OPEN I-O STOCK.                                          067

TRAITEMENT.                                                 068
   DISPLAY "1 NOUVEAU CLIENT".                              069
   DISPLAY "2 NOUVEAU STOCK".                               070
   DISPLAY "3 NOUVELLE COMMANDE".                           071
   DISPLAY "4 LISTE DES CLIENTS".                           072
   DISPLAY "5 LISTE DU STOCK".                              073
   DISPLAY "6 LISTE DES COMMANDES".                         074
   DISPLAY "0 FIN".                                         075
   ACCEPT CHOIX.                                            076

   IF CHOIX = 1                                             077
      PERFORM NOUV-CLI.                                     078
   IF CHOIX = 2                                             079
      PERFORM NOUV-STK.                                     080
   IF CHOIX = 3                                             081
      PERFORM NOUV-COM.                                     082
   IF CHOIX = 4                                             083
      PERFORM LIST-CLI.                                     084
   IF CHOIX = 5                                             085
      PERFORM LIST-STK.                                     086
   IF CHOIX = 6                                             087
      PERFORM LIST-COM.                                     088
```

```
CLOTURE.                                                        089
   CLOSE CLIENT.                                                090
   CLOSE COMMANDE.                                              091
   CLOSE STOCK.                                                 092

NOUV-CLI.                                                       093
   DISPLAY "NOUVEAU CLIENT :".                                  094
   DISPLAY "CODE DU CLIENT ?"  WITH NO ADVANCING.               095
   ACCEPT CLI-CODE.                                             096

   DISPLAY "NOM DU CLIENT : " WITH NO ADVANCING.                097
   ACCEPT NOM.                                                  098
   DISPLAY "ADRESSE DU CLIENT : " WITH NO ADVANCING.            099
   ACCEPT ADRESSE.                                              100
   DISPLAY "FONCTION DU CLIENT : " WITH NO ADVANCING.           101
   ACCEPT FONCTION.                                             102
   DISPLAY "DATE : " WITH NO ADVANCING.                         103
   ACCEPT DATE-ENREG.                                           104
   MOVE SIGNALETIQUE TO CLI-SIGNAL.                             105
   DISPLAY CLI-SIGNAL.                                          106
   PERFORM INIT-HISTO.                                          107
   WRITE CLI                                                    108
     INVALID KEY DISPLAY "ERREUR".                              109

LIST-CLI.                                                       110
   DISPLAY "LISTE DES CLIENTS".                                 111
   CLOSE CLIENT.                                                112
   OPEN I-O CLIENT.                                             113
   MOVE 1 TO END-FILE.                                          114
   PERFORM LIRE-CLI UNTIL (END-FILE = 0).                       115

LIRE-CLI.                                                       116
   READ CLIENT NEXT                                             117
      AT END MOVE 0 TO END-FILE                                 118
      NOT AT END                                                119
         DISPLAY CLI-CODE                                       120
         DISPLAY CLI-SIGNAL                                     121
         DISPLAY CLI-HISTORIQUE.                                122

NOUV-STK.                                                       123
   DISPLAY "NOUVEAU STOCK".                                     124
   DISPLAY "NUM PRODUIT : " WITH NO ADVANCING.                  125
   ACCEPT STK-CODE.                                             126

   DISPLAY "LIBELLE : " WITH NO ADVANCING.                      127
   ACCEPT STK-LIBELLE.                                          128

   DISPLAY "NIVEAU : " WITH NO ADVANCING.                       129
   ACCEPT STK-NIVEAU.                                           130

   WRITE STK                                                    131
     INVALID KEY DISPLAY "ERREUR ".                             132
```

```
LIST-STK.                                                          133
   DISPLAY "LISTE DU STOCK ".                                      134

   CLOSE STOCK.                                                    135
   OPEN I-O STOCK.                                                 136

   MOVE 1 TO END-FILE.                                             137
   PERFORM LIRE-STK UNTIL END-FILE = 0.                            138

LIRE-STK.                                                          139
   READ STOCK NEXT                                                 140
      AT END MOVE 0 TO END-FILE                                    141
      NOT AT END                                                   142
         DISPLAY STK-CODE                                          143
         DISPLAY STK-LIBELLE                                       144
         DISPLAY STK-NIVEAU.                                       145

NOUV-COM.                                                          146
   DISPLAY "NOUVELLE COMMANDE".                                    147
   DISPLAY "NUM COMMANDE : " WITH NO ADVANCING.                    148
   ACCEPT COM-CODE.                                                149

   MOVE 1 TO END-FILE.                                             150
   PERFORM LIRE-CODE-CLI UNTIL END-FILE = 0.                       151
   MOVE CLI-SIGNAL TO SIGNALETIQUE.                                152
   DISPLAY NOM.                                                    153
   MOVE CLI-CODE TO COM-CLIENT.                                    154
   MOVE CLI-HISTORIQUE TO LIST-ACHAT.                              155

   SET IND-DET TO 1.                                               156
   MOVE 1 TO END-FILE.                                             157
   PERFORM LIRE-DETAIL UNTIL END-FILE = 0 OR IND-DET = 21.         158
   MOVE LIST-DETAIL TO COM-DETAIL.                                 159

   WRITE COM                                                       160
     INVALID KEY DISPLAY "ERREUR".                                 161

   MOVE LIST-ACHAT TO CLI-HISTORIQUE.                              162
   REWRITE CLI                                                     163
     INVALID KEY DISPLAY "ERREUR CLI".                             164

LIRE-CODE-CLI.                                                     165
   DISPLAY "NUM DU CLIENT : " WITH NO ADVANCING.                   166
   ACCEPT CLI-CODE.                                                167
   MOVE 0 TO END-FILE.                                             168
   READ CLIENT INVALID KEY                                         169
      DISPLAY "CLIENT INEXITANT"                                   170
      MOVE 1 TO END-FILE                                           171
   END-READ.                                                       172

LIRE-DETAIL.                                                       173
   DISPLAY "CODE DU PRODUIT (0 = FIN) : ".                         174
   ACCEPT CODE-PROD.                                               175
   IF CODE-PROD = "0"                                              176
      MOVE 0 TO END-FILE                                           177
      MOVE 0 TO REF-STK-DE(IND-DET)                                178
   ELSE                                                            179
      PERFORM LIRE-CODE-PROD.                                      180
```

```
LIRE-CODE-PROD.                                                  181
   MOVE 1 TO EXIST-PROD.                                         182
   MOVE CODE-PROD TO STK-CODE.                                   183
   READ STOCK INVALID KEY                                        184
      MOVE 0 TO EXIST-PROD.                                      185
   IF EXIST-PROD = 0                                             186
      DISPLAY "PRODUIT INEXISTANT"                               187
   ELSE                                                          188
      PERFORM MAJ-COM-DETAIL.                                    189

MAJ-COM-DETAIL.                                                  190
   MOVE 1 TO NEXT-DET.                                           191
   DISPLAY "QUANTITE COMMANDEE : " WITH NO ADVANCING.            192
   ACCEPT Q-COM(IND-DET).                                        193
   PERFORM UNTIL                                                 194
            REF-STK-DE(NEXT-DET) = CODE-PROD                     195
         OR IND-DET = NEXT-DET                                   196
      ADD 1 TO NEXT-DET                                          197
   END-PERFORM.                                                  198
   IF IND-DET = NEXT-DET                                         199
      MOVE CODE-PROD TO REF-STK-DE(IND-DET)                      200
      PERFORM MAJ-CLI-HISTO                                      201
      SET IND-DET UP BY 1                                        202
   ELSE                                                          203
      DISPLAY "ERREUR : PRODUIT DEJA COMMANDE".                  204

MAJ-CLI-HISTO.                                                   205
   SET IND TO 1.                                                 206
   PERFORM UNTIL                                                 207
           REF-STK-AC(IND) = CODE-PROD                           208
        OR REF-STK-AC(IND) = 0 OR IND = 101                      209
      SET IND UP BY 1                                            210
   END-PERFORM.                                                  211
   IF IND = 101                                                  212
      DISPLAY "ERREUR : HISTORIQUE TROP PETIT"                   213
   EXIT.                                                         214
   IF REF-STK-AC(IND) = CODE-PROD                                215
      ADD Q-COM(IND-DET) TO TOT(IND)                             216
   ELSE                                                          217
      MOVE CODE-PROD TO REF-STK-AC(IND)                          218
      MOVE Q-COM(IND-DET) TO TOT(IND).                           219

LIST-COM.                                                        220
   DISPLAY "LISTE DES COMMANDES ".                               221
   CLOSE COMMANDE.                                               222
   OPEN I-O COMMANDE.                                            223
   MOVE 1 TO END-FILE.                                           224
   PERFORM LIRE-COM UNTIL END-FILE = 0.                          225
```

```
LIRE-COM.                                                          226
   READ COMMANDE NEXT                                              227
      AT END MOVE 0 TO END-FILE                                    228
      NOT AT END                                                   229
         DISPLAY "COM-CODE " WITH NO ADVANCING                     230
         DISPLAY COM-CODE                                          231
         DISPLAY "COM-CLIENT " WITH NO ADVANCING                   232
         DISPLAY COM-CLIENT                                        233
         DISPLAY "COM-DETAIL "                                     234
         MOVE COM-DETAIL TO LIST-DETAIL                            235
         SET IND-DET TO 1                                          236
         MOVE 1 TO END-DETAIL                                      237
         PERFORM AFFICH-DETAIL.                                    238

INIT-HISTO.                                                        239
   SET IND TO 1.                                                   240
   PERFORM UNTIL IND = 100                                         241
      MOVE 0 TO REF-STK-AC(IND)                                    242
      MOVE 0 TO TOT(IND)                                           243
      SET IND UP BY 1                                              244
   END-PERFORM.                                                    245
   MOVE LIST-ACHAT TO CLI-HISTORIQUE.                              246

AFFICH-DETAIL.                                                     247
   MOVE 0 TO IND-DET.                                              248
   IF IND-DET = 21                                                 249
      MOVE 0 TO END-DETAIL                                         250
      EXIT.                                                        251
   IF REF-STK-DE(IND-DET) = 0                                      252
      MOVE 0 TO END-DETAIL                                         253
   ELSE                                                            254
      DISPLAY REF-STK-DE(IND-DET)                                  255
      DISPLAY Q-COM(IND-DET)                                       256
      SET IND-DET UP BY 1.                                         257
   MOVE IND-DET TO IND.                                            258
   DISPLAY IND.                                                    259
```

A23.9.9 Complément - Le code SQL-DDL de la base de données relationnelle

```
create database CLICOM;


create dbspace SP_COMMANDE;
create dbspace SP_STOCK;
create dbspace SP_CLIENT;


create table CLIENT (
     CODE_CLIENT   char(12) not null,
     NOM           char(20) not null,
     ADRESSE       char(40) not null,
     FONCTION      char(10) not null,
     DATE_ENREG    char(10) not null,
     primary key (CODE_CLIENT))
     in SP_CLIENT;

create table STOCK (
     CODE_STOCK    numeric(5) not null,
     LIBELLE       char(100)  not null,
     NIVEAU        numeric(5) not null,
     primary key (CODE_STOCK))
     in SP_STOCK;

create table COMMANDE (
     CODE_COMMANDE numeric(10) not null,
     CODE_CLIENT   char(12)    not null,
     primary key (CODE_COMMANDE))
     in SP_COMMANDE;

create table DETAIL (
     CODE_COMMANDE numeric(10) not null,
     NUM_DETAIL    numeric(2)  not null,
     CODE_STOCK    numeric(5)  not null,
     Q_COM         numeric(5)  not null,
     primary key (CODE_COMMANDE, NUM_DETAIL))
     in SP_COMMANDE;

create table ACHAT (
     CODE_STOCK   numeric(5) not null,
     CODE_CLIENT  char(12)   not null,
     TOTAL        numeric(5) not null,
     primary key (CODE_CLIENT, CODE_STOCK))
     in SP_STOCK;


alter table ACHAT add constraint FKACH_CLI
     foreign key (CODE_CLIENT) references CLIENT;

alter table ACHAT add constraint FKACH_STO
     foreign key (CODE_STOCK) references STOCK;

alter table COMMANDE add constraint FKPASSE
     foreign key (CODE_CLIENT) references CLIENT;
```

```
alter table DETAIL add constraint FKREFERENCE
     foreign key (CODE_STOCK) references STOCK;

alter table DETAIL add constraint FKDE|1
     foreign key (CODE_COMMANDE) references COMMANDE;


create index FKACH_STO
     on ACHAT (CODE_STOCK);

create index FKPASSE
     on COMMANDE (CODE_CLIENT);

create index FKREFERENCE
     on DETAIL (CODE_STOCK);

-- index implicite
-- create unique index CLI_CODE
--     on CLIENT (CODE_CLIENT);

-- index implicite
-- create unique index STK_CODE
--      on STOCK (CODE_STOCK);

-- index implicite
-- create unique index COM_CODE
--      on COMMANDE (CODE_COMMANDE);

-- index implicite
-- create unique index IDSEC
--      on DETAIL (CODE_COMMANDE, NUM_DETAIL);

-- index implicite
-- create unique index IDACHAT
--     on ACHAT (CODE_CLIENT, CODE_STOCK);


create exception DETAIL_CARD_20 "Erreur: pas plus de 20 DETAILs
       par COMMANDE";
create exception DETAIL_SEC_ID  "Erreur: {CODE_COMMANDE,
       CODE_STOCK} identifiant de DETAIL";


create trigger tr_DETAIL_CI_ins
before insert on DETAIL
for each row
declare N integer;
begin
   -- contrainte de cardinalite [0-20]
   select count(*) into N from DETAIL
   where  CODE_COMMANDE = new.CODE_COMMANDE;
   if (N = 20) then exception DETAIL_CARD_20;
   -- identifiant secondaire
   select count(*) into N from DETAIL
   where  CODE_COMMANDE = new.CODE_COMMANDE
   and    CODE_STOCK    = new.CODE_STOCK;
   if (N = 1)  then exception DETAIL_SEC_ID;
end;
```

```
create trigger tr_DETAIL_CI_upd
before update of CODE_COMMANDE,NUM_DETAIL,CODE_STOCK on DETAIL
for each row
declare N integer;
begin
   -- contrainte de cardinalite [0-20]
   if (new.CODE_COMMANDE <> old.CODE_COMMANDE)
   then begin
      select count(*) into N from DETAIL
      where  CODE_COMMANDE = new.CODE_COMMANDE;
      if (N = 20) then exception DETAIL_CARD_20;
   end;
   -- identifiant secondaire
   if (new.CODE_COMMANDE <> old.CODE_COMMANDE
       or new.CODE_STOCK <> old.CODE_STOCK)
   then begin
      select count(*) into N from DETAIL
      where  CODE_COMMANDE = new.CODE_COMMANDE
      and    CODE_STOCK    = new.CODE_STOCK;
      if (N = 1)  then exception DETAIL_SEC_ID;
   end;
end;
```

A23.10DICTIONNAIRE DE DONNÉES

Les catalogues des SGBD relationnels comprennent des tables qui décrivent les schémas des bases de données accessibles. C'est ainsi qu'il existe une *table des tables*, une *table des colonnes*, une *table des utilisateurs*, une *table des privilèges*, ainsi qu'une vingtaine d'autre tables, parmi lesquelles une table qui décrit les composants des clés de chaque base de données. La table de la A23.10.1 en est un exemple. Nous ne disposons que de cette représentation graphique, supposée extraite de la documentation du fournisseur du SGBD. Les éventuels identifiants, clés étrangères et autres contraintes relatifs à cette table sont implicites. On ne dispose d'aucune autre information que sa structure et son contenu à un instant déterminé. Nous allons tenter de reconstruire le schéma conceptuel de cette table.

SYSKEY					
CompID	**TableName**	**ColumnName**	**KeyType**	**KeyID**	**RefComp**
1	CLIENT	NCLI	P	1	
2	COMMANDE	NCOM	P	2	
3	COMMANDE	NCLI	F	3	1
4	PRODUIT	NPRO	P	4	
5	DETAIL	NCOM	P	5	
6	DETAIL	NPRO	P	5	
7	DETAIL	NCOM	F	6	2
8	DETAIL	NPRO	F	7	4
9	EXPEDITION	NCOM	P	8	
10	EXPEDITION	NPRO	P	8	
11	EXPEDITION	DATE	P	8	
12	EXPEDITION	NCOM	F	9	5
13	EXPEDITION	NPRO	F	9	6
14	FACTURE	NFACT	P	10	
15	FACTURE	NCLI	F	11	1

Figure A23.10.1 - Table du catalogue d'un SGBD décrivant les clés d'une base de données.

A23.10.1Remarques préliminaires

1. Les tables du catalogue constituent une ressource critique lors de l'exécution des requêtes. Il est logique que leur structure soit fortement optimisée pour la consultation. C'est la principale raison de l'absence de contraintes d'intégrité explicites. Il faut donc s'attendre aussi à y trouver des redondances, et en particulier des structures dénormalisées.
2. Les tables du catalogue sont mises à la disposition des programmeurs d'application. On peut donc légitimement supposer que les noms des colonnes ont été choisis avec un certain soin, de manière à évoquer la fonction de celles-ci.
3. Le contenu des tables du catalogue est géré par le SGBD. On peut admettre que ces tables ne comportent aucune erreur.
4. Le contenu des tables du catalogue est l'image du schéma d'une base de données dont on connaît normalement la structure. Il est donc possible, si nécessaire, de comparer ce contenu à ce schéma.

A23.10.2Méthodologie

Nous adopterons la méthode classique comportant trois étapes : extraction physique, reconstruction du schéma logique et conceptualisation.

A23.10.3Extraction physique

Nous ne disposons pas du code DDL relatif à cette table. Par simple inspection de la table SYSKEY, nous tirons le schéma physique de la A23.10.2.

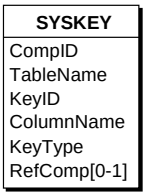


Figure A23.10.2 - Structure initiale de la table SYSKEY.

A23.10.4Reconstruction du schéma logique

La phase de reconstruction du schéma logique est cruciale en raison du manque de déclarations explicites de contraintes et de la présence probable de structures d'optimisation. Pour cette extraction, nous pouvons nous baser sur trois sources d'information : les noms, le contenu de la table et notre connaissance du domaine d'application (les schémas relationnels).

a) Analyse des noms

Nous exploitons l'hypothèse d'un choix raisonné et systématique des noms de colonnes.

1. Deux colonnes ont un nom qui suggère un rôle d'identifiant : CompID et KeyID
2. La colonne (facultative) RefComp possède un nom formé d'un composant fonctionnel (Ref) et d'un composant structurel (Comp), déjà rencontré dans CompID. On peut penser à une clé étrangère vers une table représentant des objets dont le type est désigné par "Comp".

b) Analyse des données

• Identifiants (1)

En considérant chaque colonne individuellement, on observe immédiatement deux faits :

- les valeurs de la colonne CompID sont distinctes,

- aucune autre colonne ne jouit de cette propriété.

Le premier fait peut être évalué de manière plus précise par la requête SQL suivante, qui doit renvoyer une réponse vide (une recherche systématique des identifiants réclamerait 6 requêtes de ce type) :

```
select count(*) from SYSKEY
group by CompID
having count(*) > 1
```

Hypothèse : {CompID} est un identifiant de SYSKEY

• Identifiants (2)

Considérant les colonnes obligatoires deux par deux, on observe que le groupe {KeyID, ColumnName} possède des valeurs distinctes. Il n'y a pas d'autre groupes qui présentent cette propriété. A valider par une requête SQL comme nous l'avons fait ci-dessus. Une recherche systématique des identifiants minimaux à deux composants réclamerait 24 (= 30 moins ceux qui incluent CompID) requêtes. Le développement d'un petit générateur de requêtes serait utile pour des tables plus importantes.

Hypothèse : {KeyID, ColumnName} est un identifiant de SYSKEY

• Dépendances fonctionnelles

L'hypothèse de forte optimisation en temps de consultation rend réaliste une analyse poussée des dépendances fonctionnelles. En l'absence des requêtes de mise à jour, qui auraient pu nous donner des indices sur ces DF, il nous reste à analyser les données.

Une inspection visuelle est illusoire dans des situations plus complexes que celle-ci. D'autre part, une analyse systématique va s'avérer très lourde et nécessitera le développement d'un générateur de requêtes. En effet, un calcul rapide montre qu'une table de 6 colonnes peut être le siège de 186 dépendances fonctionnelles :

DF du type A $\longrightarrow$ F : 6x5 = 30
DF du type AB $\longrightarrow$ F : 6x5x4/2 = 60
DF du type ABC $\longrightarrow$ F : 6x5x4x3/(3x2) = 60
DF du type ABCD $\longrightarrow$ F : 6x5x4x3x2/(4x3x2) = 30
DF du type ABCDE $\longrightarrow$ F : 6x5x4x3x2/(5x4x3x2) = 6

Nous réduirons à 24 le nombre de DF potentielles à évaluer :

- on examine les DF dont le déterminant comprend une seule colonne obligatoire, à l'exclusion des identifiants; ceci concerne les 4 colonnes TableName, ColumnName, KeyType et KeyID, chacune associée à trois déterminés potentiels, soit 12 DF potentielles;
- on examine les DF dont le déterminant comprend deux colonnes obligatoires, à l'exclusion des identifiants et des déterminants déjà trouvés; ceci concerne 6

déterminants au maximum, chacun associée à deux déterminés potentiels, soit 12 DF potentielles;

L'évaluation de la DF R(A,B,C): A ⟶ B peut s'appuyer sur la requête suivante , qui doit renvoyer une réponse vide :

```
create view S(A,B) as select distinct A,B from R;
select A, count(*) from S group by A having count(*) > 1
```

ou encore, de manière plus concise :

```
select A, count(distinct B) from R
group by A having count(distinct B) > 1
```

L'analyse de la table SYSKEY nous permet de formuler les hypothèses suivantes :

Hypothèses :KeyID ⟶ KeyType
KeyID ⟶ TableName

• Contraintes d'inclusion

La colonne RefComp pourrait bien être une clé étrangère. L'analyse de ses valeurs montre que si la table cible est SYSKEY, alors les hypothèses suivantes sont plausibles :

Hypothèses :SYSKEY[RefComp] ⊆ SYSKEY[CompID]
SYSKEY[RefComp] ⊆ SYSKEY[KeyID]

• Autres propriétés

Le domaine de KeyType semble être {'P', 'F'}, ce qu'on vérifie aisément par la requête :

```
select distinct KeyType from SYSKEY
```

On observe que pour toute ligne telle que RefComp est valuée, alors KeyType = 'F', et inversement Si cette hypothèse se confirme, il s'agit d'une redondance structurelle.

c) Conclusions

Après réflexion, on retient les structures suivantes :

1. **Identifiants**. La table SYSKEY contient deux identifiants :
   {CompID},
   {KeyID,ColumnName}.
2. **Dépendances fonctionnelles**. La table SYSKEY contient deux DF non issues des identifiants :
   KeyID ⟶ KeyType
   KeyID ⟶ TableName

3. **Contraintes d'inclusion**. De l'analyse des noms et de celle des données, on retire :

   SYSKEY[RefComp] $\subseteq$ SYSKEY[CompID]

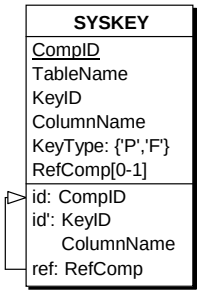
4. **Autres propriétés**.

   dom(KeyType): {'P','F'}.

   s.RefComp $\neq$ null $\Leftrightarrow$ s.KeyType = 'F'

5. **Autres propriétés (bis)**. Ayant admis que {RefComp} est une clé étrangère vers SYSKEY, on observe que toutes les lignes référencées ont une valeur KeyType = 'P' .

Le schéma enrichi se présente comme suit :



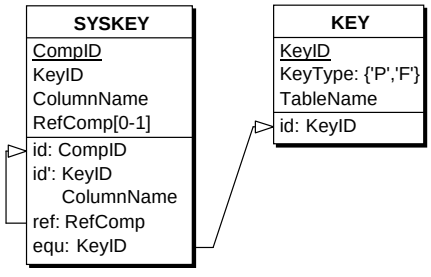
KeyID ⟶ KeyType,TableName
corrélation entre FK et KeyType

Figure A23.10.3 - Schéma logique complet de la table SYSKEY.

A23.10.5CONCEPTUALISATION DES STRUCTURES DE DONNEES

a) Désoptimisation

La colonne KeyID est un identifiant mais n'est pas un identifiant de la table. Celle-ci n'est donc pas normalisée. Sa décomposition donne le schéma A23.10.4.



corrélation entre FK et KeyType

Figure A23.10.4 - Normalisation selon la DF KeyID ⟶ KeyType,TableName

b) Conceptualisation

Les clés étrangères sont transformées en types d'associations (A23.10.5).

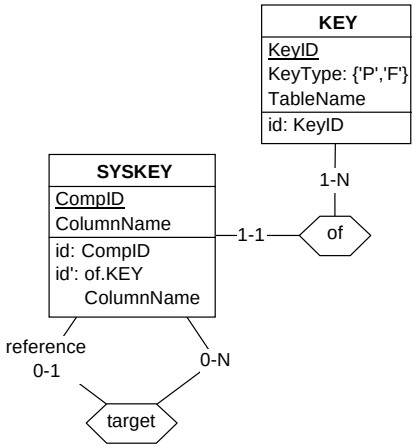


Figure A23.10.5 - Les clés étrangères sont transformées en types d'associations.

c) Normalisation

Le schéma conceptuel obtenu à l'étape précédente doit encore faire l'objet d'une évaluation de normalisation.

- Le type d'entités SYSKEY représente les composants des KEYs. On le renomme COMPONENT.
- Il est utile de mettre en évidence les deux types de clés : primary, nommé P_KEY (KeyType = 'P') et foreign, nommé F_KEY (KeyType = 'F'). Nous faisons de même pour leurs composants P_COMPONENT et F_COMPONENT. Il est ainsi possible de décrire de manière structurelle le fait que les composants de type F référencent, via target, des composants de type P. L'attribut KeyID, devenu inutile, est supprimé. L'existence de circuits suggère des contraintes liées à ceux-ci (à déterminer).

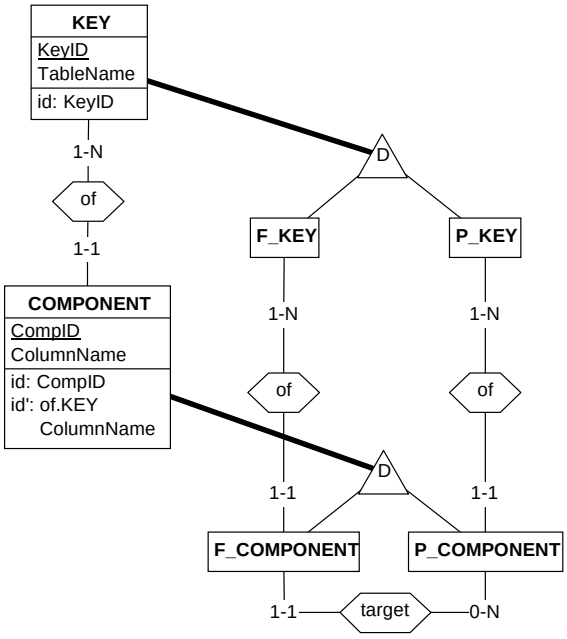


Figure A23.10.6 - Schéma conceptuel dérivé des propriétés de la table SYSKEY.

- De la connaissance du domaine d'application nous tirons l'existence des concepts centraux de table et de colonne. Les attributs TableName et ColumnName qui représentent ces concepts sont transformés en types d'entités. On déterminera les contraintes liées aux circuits.

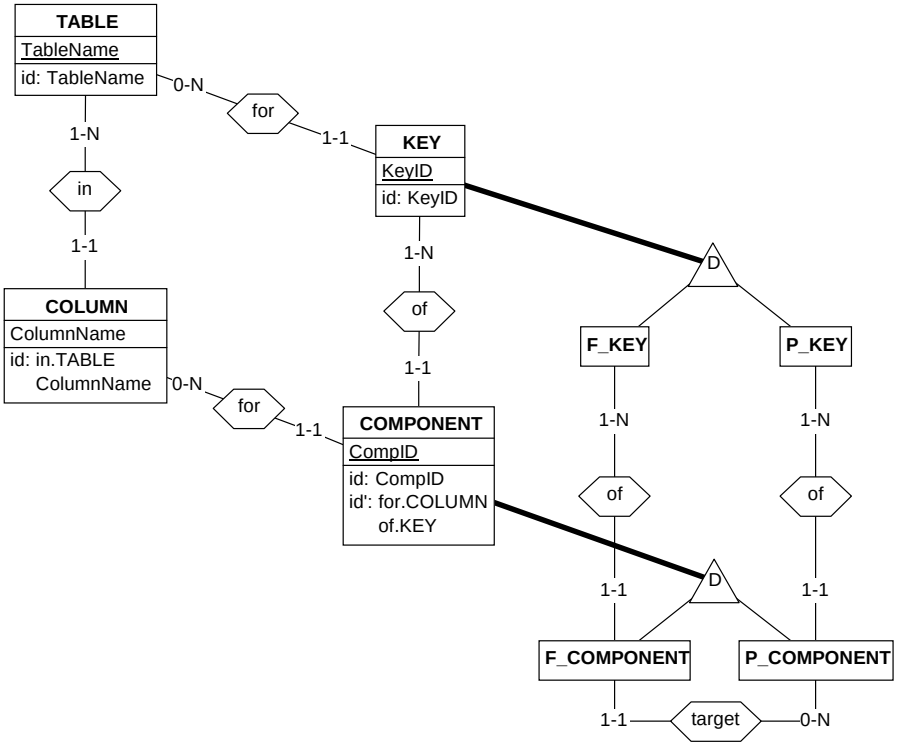


Figure A23.10.7 - Schéma conceptuel de la table SYSKEY mettant en évidence les concepts de table et de colonne.

A23.11ANALYSE D'UN STACK HYPERCARD

A23.11.1Présentation

On récupère une petite application personnelle destinée à gérer sur un Macintosh une base de données bibliographique scientifique. On désire convertir cette application sous une forme relationnelle ou orientée-objet. On demande donc avant toute migration de reconstituer le schéma conceptuel de cette base de données.

L'application elle-même et son code étant disponibles et documentés[22], la tâche ne devrait pas être considérable, d'autant plus que nous nous baserons sur une version quelque peu simplifiée de cette application. Son intérêt réside essentielle-

22. Le code HyperTalk de l'application est disponible sur demande.

ment dans l'application d'une technique de factorisation d'attributs permettant de reconstruire une hiérarchie *is-a*.

Hypercard est un environnement qui fut très populaire sur Macintosh entre 1987 et 2004[23]. Très puissant et d'une simplicité remarquable, il permettait de construire d'une manière intuitive de petites applications de bases de données. Bien qu'étant lui-même orienté-objet, cet environnement ne permettait pas de développer des applications orientées-objet, ce qui en revanche fera l'intérêt de cette étude. Très sommairement, les concepts d'Hypercard qui nous seront utiles sont les suivants. Une application se dénomme un *stack*, et se présente comme une pile de cartes. Chaque carte est un enregistrement dérivé d'un modèle, qui est lui-même une carte (comme dans les modèles de *frames*). Celui-ci est constitué de champs et de boutons sur un fond passif pouvant comporter dessins ou libellés, mais aussi champs et boutons. Les événements classiques (souris, clavier, création, suppression d'une carte, modification d'un champ, ouverture, fermeture d'un stack, etc) peuvent être captés par des méthodes attachées aux différents objets de la hiérarchie de l'application : boutons, champs, carte, fond, modèle, stack et Hypercard. Ces méthodes sont rédigées dans un langage pseudo-naturel, dénommé HyperTalk, donnant accès à toutes les ressources de l'application, mais aussi du système Macintosh.

Nous utiliserons explicitement les concepts de *stack* (base de données), de *modèle de carte* (type d'enregistrements) et de *champs* mono- ou multivalué. Nous ferons à l'occasion appel à des indications externes, telles que des contraintes d'intégrité, qui auront été formulées à partir de l'analyse du code de gestion des données, mais que nous ne justifierons pas pour ne pas alourdir l'exposé.

A23.11.2Extraction du schéma physique brut

L'application fait usage de huit modèles de cartes dont les champs sont représentés à la A23.11.1 Ce schéma est obtenu par l'examen visuel de ces modèles et la reprise des noms des champs. A l'exception du champ commun IDdoc, qui semble constituer une sorte d'identifiant global des cartes, tous les champs sont a priori facultatifs.

23. Il est réputé avoir inspiré plusieurs techniques populaires aujourd'hui telles que http, JavaScript, Wiki et ... les macro-virus !
Voir *http://en.wikipedia.org/wiki/HyperCard* et *http://en.allexperts.com/e/h/hy/hypercard.htm*

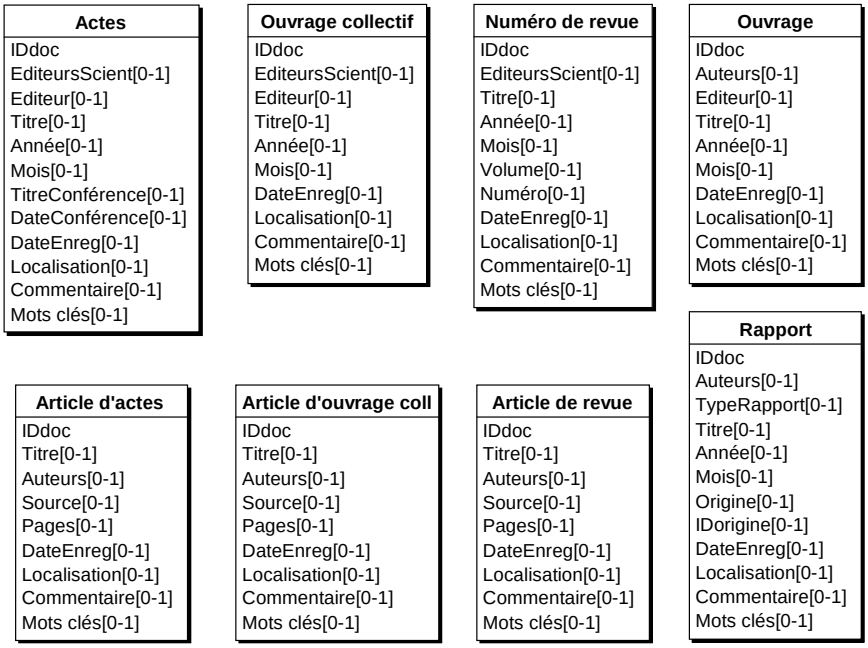


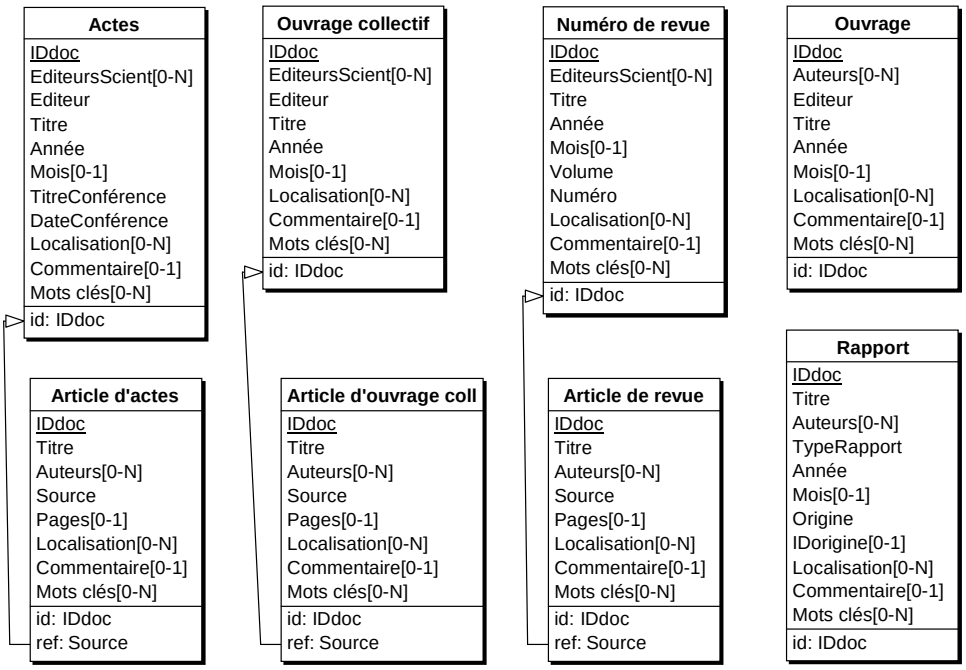
Figure A23.11.1 - Schéma physique brut

A23.11.3Affinement du schéma physique

L'analyse des sources d'information liées à l'application (code source, documentation, aide en ligne, exécution) nous donne une quantité importante d'indications complémentaires sur les propriétés des champs. Nous en retiendrons trois :

1. IDdoc est effectivement un champ identifiant les enregistrements indépendamment de leur type. Ils s'agit donc d'un *identifiant global*.
2. Les méthodes de saisie et de modification des enregistrements montrent que certains champs sont *obligatoires*.
3. Pour certains champs dont la valeur peut être formée de plusieurs lignes de texte, il apparaît que chaque ligne est considérée comme une valeur individuelle. On déclarera ces champs *multivalués* de cardinalité indéfinie [0-N].
4. Les trois champs intitulés Source sont systématiquement utilisés pour accéder à un autre enregistrement. Ils constituent donc des *clés étrangères*.

Ces informations complémentaires sont consignées dans le schéma de la A23.11.2.



id(Actes, Ouvrages collectifs, ..., Rapport): IDdoc[24]

Figure A23.11.2 - Schéma physique affiné

A23.11.4Première conceptualisation (clés étrangères, attributs composés)

La phase de conceptualisation, qui a pour objectif la reconstruction d'un schéma conceptuel plausible, inclut l'interprétation des clés étrangères comme des types d'associations. On détecte également la trace de la désagrégation d'un attribut composé (Conférence). Le schéma de la A23.11.2. montre le résultat de la phase de détraduction du processus de conceptualisation.

24. Notation simplifiée dans le cas où les composants de l'identifiant portent le même nom dans tous les types d'entités.

A23.11.5Factorisation des attributs

On observe rapidement que des attributs de même nom apparaissent dans plusieurs types d'entités. Ce phénomène pourrait être accidentel et dériver d'une simple homonymie.

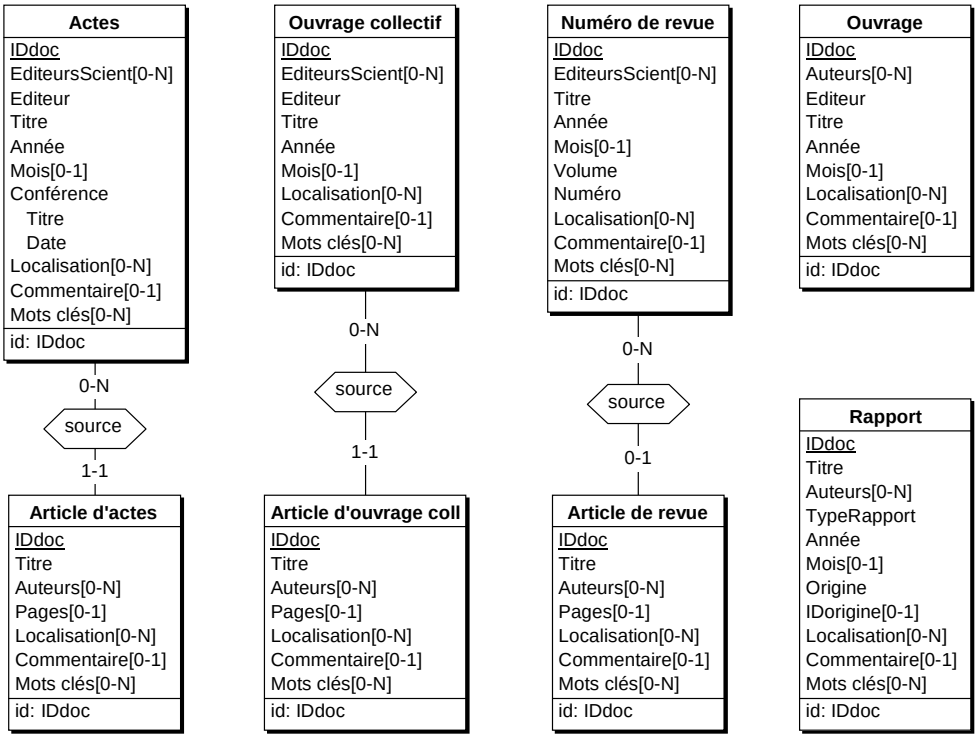


Figure A23.11.3 - Premier schéma conceptuel - Expression des types d'associations et des attributs composés.

Il pourrait en revanche signifier que ces types d'entités se partagent effectivement ces attributs communs, et que ce partage représente une proximité sémantique significative : deux types d'entités E1 et E2 qui possèdent un même attribut A relèvent tous deux d'une catégorie d'ordre supérieur EA, celle des entités dotées de A. Ce phénomène n'est rien d'autre que la mise en évidence d'un **surtype** commun éventuel[25] à E1 et E2. Nous avons déjà rencontré ce pattern à la section figure F.17.7.2/f (*Explicitation des constructions insuffisamment expressives* / *Relations is-a implicites*). Ce schéma n'est donc pas normalisé.

On vérifie qu'à un même nom correspondent des attributs de même sémantique, et jouant des rôles identiques au sein de leurs types d'entités. Faisons l'hypothèse que notre schéma satisfait cette hypothèse. Par exemple, l'attribut Auteurs, partout

25. C'est-à-dire qui reste à valider sémantiquement.

où il apparaît, désigne les personnes qui ont rédigé le document, quelle qu'en soit la forme.

a) Construction de la matrice des attributs

La technique que nous allons employer, dérivée de celle des *treillis de Galois*[26], est basée sur une matrice matérialisant la relation binaire R(E,A) indiquant, pour tout couple (e,a) de R, que le type d'entités e possède l'attribut a.

Nous proposerons un algorithme intuitif facile à appliquer manuellement pour des schémas de petites tailles[27]. Comme nous serons amenés à créer des noms par concaténation de noms de types d'entités existants, nous renommerons de manière plus concise les types d'entités sources :

```
Actes                    ⟶   ACTE
Article d'actes          ⟶   AACT
Ouvrage collectif        ⟶   COLL
Article d'ouvrage coll   ⟶   ACOL
Numéro de revue          ⟶   REVU
Ouvrage                  ⟶   OUVR
Article de revue         ⟶   AREV
Rapport                  ⟶   RAPP
```

Construisons un tableau reprenant tous les types d'attributs et tous les attributs, et indiquant pour chaque couple si cet attribut apparaît dans le type d'entités. Ce tableau se présente comme à la A23.11.4.

	ACTE	AACT	COLL	ACOL	REVU	OUVR	AREV	RAPP
Année	X		X		X	X		X
Auteurs		X		X		X	X	X
Commentaire	X	X	X	X	X	X	X	X
Conférence	X							
Editeur	X		X			X		
EditeursScient	X		X		X			
IDdoc	X	X	X	X	X	X	X	X
IDorigine								X
Localisation	X	X	X	X	X	X	X	X
Mois	X		X		X	X		X
MotsClés	X	X	X	X	X	X	X	X
Numéro					X			
Origine								X
Pages		X		X			X	
Titre	X	X	X	X	X	X	X	X

26. Voir par exemple : Gammoudi, M., Nafkha, I., A Formal method for inheritance graph hierarchy construction, *Information Sciences*, 140 (2002), pp. 295-317, Elsevier

27. Pour des schémas plus importants, on pourra utiliser un script SQL disponible chez l'auteur.

TypeRapport								X
Volume					X			

Figure A23.11.4 - Tableau des attributs du schéma de la A23.11.3.

Nous trions ce tableau de manière que les lignes de même contenu (chaîne de X) soient consécutives (A23.11.5). L'ordre des groupes de lignes identiques est indifférent.

	ACTE	AACT	COLL	ACOL	REVU	OUVR	AREV	RAPP
IDdoc	X	X	X	X	X	X	X	X
Titre	X	X	X	X	X	X	X	X
Localisation	X	X	X	X	X	X	X	X
Commentaire	X	X	X	X	X	X	X	X
MotsClés	X	X	X	X	X	X	X	X
Année	X		X		X	X		X
Mois	X		X		X	X		X
Auteurs		X		X		X	X	X
Editeur	X		X			X		
EditeursScient	X		X		X			
Pages		X		X			X	
IDorigine								X
TypeRapport								X
Origine								X
Numéro					X			
Volume					X			
Conférence	X							

Figure A23.11.5 - Tableau des attributs trié par agrégation des lignes identiques. On met en évidence neuf groupes de lignes.

b) Construction des types d'entités et distribution des attributs

Chaque groupe de lignes identiques constitue un agrégat que nous transformons en un type d'entités de la manière suivante :

1. son nom est formé de la concaténation des noms de colonnes où un X apparaît; nous chercherons plus tard à lui donner un nom significatif;
2. ses attributs sont ceux des lignes de l'agrégat.

En outre, pour chaque type d'entités source n'apparaissant pas encore dans le schéma qui vient d'être élaboré, on ajoute un type d'entités portant son nom mais sans attributs.

On obtient ainsi le schéma de la A23.11.6. Pour alléger la disposition, le type d'entités dont toutes les lignes sont à X est nommé ACTE_AACT_..._AREV_RAPP.

c) Création de la hiérarchie ISA

Nous allons ensuite créer parmi ces types d'entités la hiérarchie *is-a* qui par héritage produit les types d'entités sources de la A23.11.3. Nous procédons de la manière suivante.

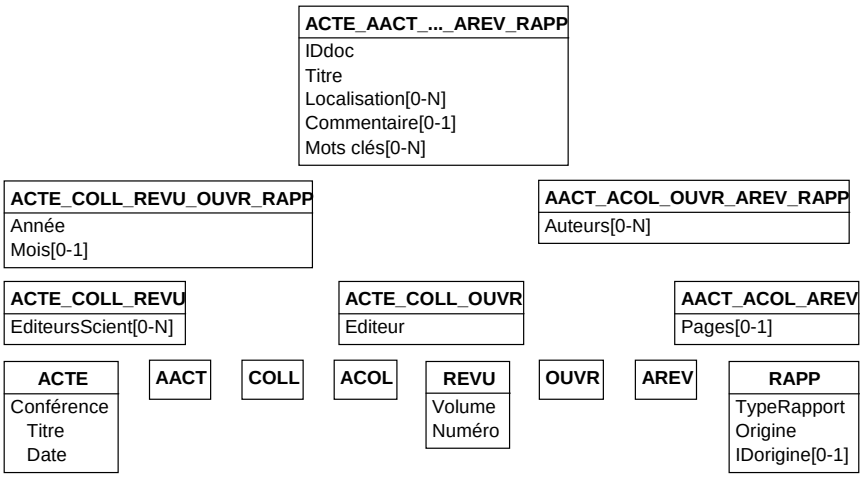


Figure A23.11.6 - Les types d'entités du futur schéma conceptuel normalisé

1. Pour tout couple de types d'entités <E1, E2> tel que le nom de E1 est entièrement inclus dans celui de E2, on déclare que *E1 est un sous-type de E2*.
2. On ne retient pas les relations *is-a* transitives.

Concrètement, on reconnaît qu'un nom *est inclus* dans un autre si tous les fragments constitutifs du premier sont présents dans le second. Ainsi, on établit que

"**ACTE_COLL_OUVR**" ⊂ "**ACTE_COLL_**REVU_**OUVR**_RAPP".

Une relation *is-a* est *transitive* si elle dérive de deux autres relations. Par exemple, on ne retiendra pas la relation *A is-a C* si les relations *A is-a B* et *B is-a C* sont établies.

On obtient ainsi le schéma de la A23.11.7.

d) Renommage des surtypes

Les conventions de dénomination des surtypes ont servi à établir les relations *is-a* par une simple analyse lexicale des noms. Il convient à présent de remplacer ces noms désormais inutiles par d'autres, sémantiquement pertinents. Cette phase est importante, car elle revient à assigner une sémantique aux concepts qui viennent d'être mis en évidence.

Commençons par rendre aux types d'entités sources leur nom d'origine. Nous devons ensuite interpréter chaque surtype en terme des concepts du domaine d'application.

- ACTE_AACT_..._AREV_RAPP : couvrant la totalité du fonds bibliographique, on peut sans risque appeler ce type d'entités Document.
- AACT_ACOL_OUVR_AREV_RAPP : son unique attribut Auteur désigne ce type d'entités comme représentant les documents ayant un ou plusieurs auteurs. Appelons-le Document à auteur(s).
- ACTE_COLL_REVU_OUVR_RAPP : malgré un appel désespéré à notre imagination, nous n'arrivons pas à trouver un nom satisfaisant pour ce type d'entités, qui ne semble pas correspondre à un concept évident. Nous lui conservons en attendant l'inspiration son nom de guerre.
- ACTE_COLL_REVU : ses sous-types correspondant à des documents regroupant d'autres documents (des articles), nous lui donnons le nom de Document collectif.
- ACTE_COLL_OUVR : considérant ses sous-types, ce type d'entités représente des livres, d'où le nom Livre.
- AACT_ACOL_AREV : on peut sans hésitation lui donner le nom d'Article.

Ces décisions conduisent au schéma de la A23.11.8.

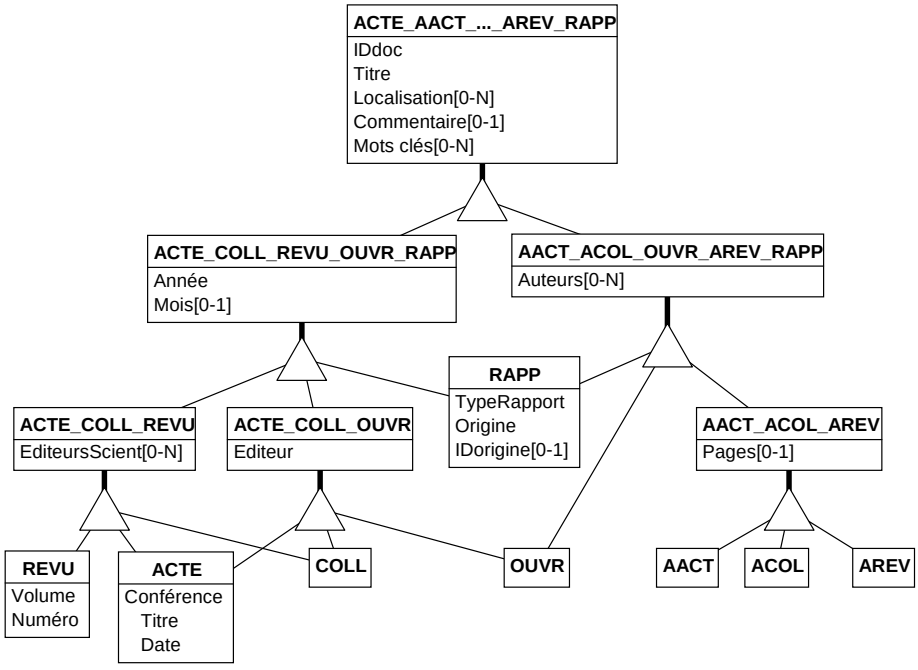


Figure A23.11.7 - Hiérarchie *is-a* construite à partir de l'analyse des noms

e) Validation sémantique et complétion du schéma

Il reste quelques questions à régler et des informations à réincorporer dans le schéma pour obtenir le schéma conceptuel final de la A23.11.9.

• Discussion des classes

Le schéma élaboré jusqu'ici pose un problème de classification : l'impossibilité de définir ACTE_COLL_REVU_OUVR_RAPP. Nous allons voir que ce problème dépend d'un autre : le faible degré d'utilité de Document à auteur(s).

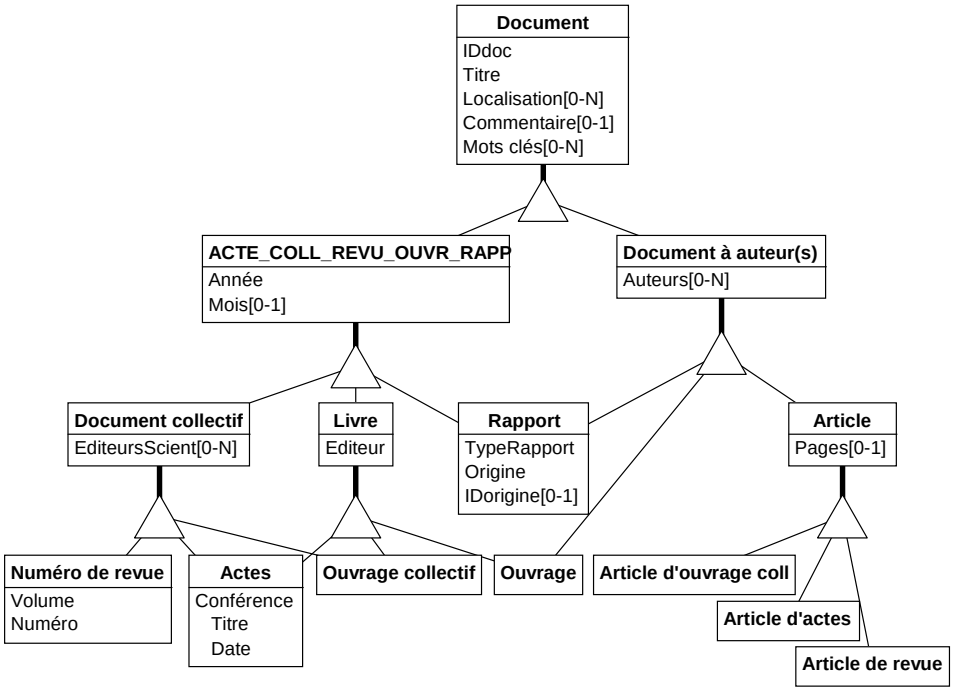


Figure A23.11.8 - Première tentative de dénomination des surtypes.

Ce dernier n'a d'autre utilité que de rassembler les *documents ayant des auteurs*, critère qu'on peut sans doute considérer comme mineur dans le contexte du domaine d'application. Si tel est notre sentiment, nous pouvons supprimer ce concept :

1. l'attribut Auteurs[0-N] est réattribué à chacun des trois sous-types Rapport, Ouvrage et Article;
2. ces trois sous-types sont rendus directement dépendants du surtype de Document à auteur(s), soit Document, sauf s'ils en dépendent déjà indirectement (pas de relations *is-a* transitives), ce qui est le cas de Rapport et Ouvrage;
3. le type d'entités Document à auteur(s) est retiré du schéma.

Le résultat de cette simplification met en lumière un fait intéressant : les sous-ensembles de sous-types de Article et de ACTE_COLL_REVU_OUVR_RAPP sont disjoints. En d'autres termes, ce dernier type d'entités regroupe tous les documents *qui ne sont pas des articles*. Parmi les documents, il y a donc la catégorie des *articles* et la catégories des *autres* (les *non-articles*). D'où le nom Autre attribué au type d'entités mystérieux. Cette classification semble suffisamment importante pour être conservée : en effet, les articles formant une classe homogène particulière des ressources du scientifique, il est donc utile aussi de représenter leur complémentaire.

• Reprise des types d'associations

Les types d'associations ont été exclus de la construction de la hiérarchie *is-a*. On peut maintenant les reprendre[28]. Soit dit en passant, il apparaît indispensable de transformer les attributs de référence (clés étrangères) avant la factorisation des attributs. Dans le cas contraire, les trois attributs Source auraient fait l'objet d'une migration vers Article, ce qui aurait entraîné une importante perte de sémantique. En résumé, on limitera cette techniques aux *attributs purs*.

• Reprise des contraintes d'intégrité

L'ancien identifiant global IDcom devient simplement l'identifiant local de Document. Ce dernier recouvrant tous les types de documents, IDcom est structurellement un **identifiant global** sans qu'il soit nécessaire de le déclarer comme tel.

• Propriétés des sous-types

L'algorithme de factorisation ne dit rien quant aux propriétés de totalité et de disjonction des ensembles de sous-types. Il faut donc réfléchir à cette question de manière spécifique. Les raisonnements doivent tenir compte du fait que le schéma conceptuel résulte, non pas d'une analyse abstraite des besoins des utilisateurs, mais d'une description physique d'une base de données existante.

Cette remarque entraîne deux conséquences importantes.

1. Les sous-types terminaux, c'est-à-dire les huit types d'entités sources, sont **disjoints**. En effet, l'application Hypercard n'autorise pas qu'on répertorie un même document selon deux modèles différents.
2. Tout surtype est construit directement ou indirectement à partir d'une collection de sous-types terminaux. Or l'application Hypercard ne permet de créer un document que s'il appartient à l'un d'eux. Tout surtype est donc soumis à une contrainte de **totalité**.

Nous rappellerons une troisième contrainte (relire la section 15.9.4) :

---

28. *Commentaire technique*. Les algorithmes standard de classification basés sur les *treillis de Galois* ne conviennent pas dans le cas présent, car ils traitent d'une manière inadéquate les types d'entités ayant le même ensemble d'attributs. En particulier, ils font disparaître les types d'entités sans attributs comme Ouvrage collectif et Article d'ouvrage coll. La variante proposée dans cette section s'impose donc, outre le fait qu'elle ne fait pas appel aux concepts complexes des techniques standards.

3. Un surtype dont au moins deux sous-types ont un sous-type en commun **ne peut être soumis à une contrainte de disjonction**. Tels est le cas de Autre, dont Document collectif et Livre se partage les sous-types Acte et Ouvrage collectif. Dans tous les autres cas, on déclarera une contrainte de disjonction.

Ces trois principes nous permettent de compléter le schéma comme suit.

- Surtype Document : totalité et disjonction, soit partition (P).
- Surtype Article : totalité et disjonction, soit partition (P).
- Surtype Autre : totalité mais pas disjonction (T).
- Surtype Document collectif : totalité et disjonction, soit partition (P).
- Surtype Livre : totalité et disjonction, soit partition (P).

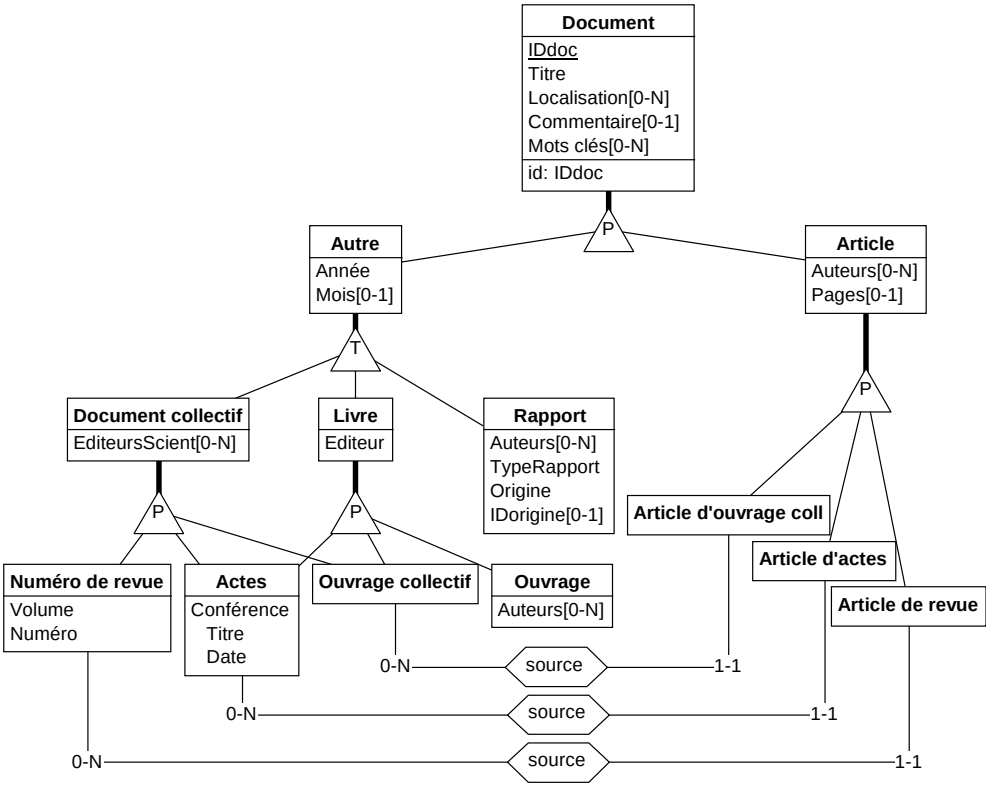


Figure A23.11.9 - Le schéma conceptuel final.

• Finalisation

Le schéma obtenu pourrait faire l'objet d'améliorations supplémentaires dont le lecteur sera seul juge :

- les attributs multivalués Editeurs-Scient, Auteurs, Localisation et Mot-clés ne pourraient-ils pas être transformés en types d'entités ?

• l'attribut Pages ne serait-il pas un identifiant d'Article relativement à sa source ?

.

A23.12SQL ET LES ONTOLOGIES

Cette section, actuellement dans un état embryonnaire, étudie différentes représentations des données, de la plus spécialisée à la plus générale. Cette dernière sera à ce point générale qu'elle n'utilisera pratiquement aucun schéma, ou, plus exactement, qu'elle utilisera le même schéma quelle que soit la base de données à représenter. C'est cette dernière représentation qui nous conduira sans difficulté à des ensembles de triplets, caractéristiques de la modélisation par les langages d'ontologies. Le format étudié porte le nom de *TripleStore* (stockage de triplets). Une référence : http://en.wikipedia.org/wiki/Triplestore.

Nous étudierons une hiérarchie de modes de représentations appliquée à notre exemple de base de données illustrée à la figure F.2.8. Les exemples de requêtes sont formulés selon le langage SQL d'Access mais leur adaptation à d'autres dialectes est immédiate[29]. Les scripts sont disponibles sur le site de l'ouvrage.

A23.12.1Les tables sources

Nous représenterons comme suit les schémas des quatre tables de la base de données sur laquelle nous travaillerons.

```
S_CLIENT  (NCLI, NOM, ADRESSE, LOCALITE, CAT, COMPTE)
S_PRODUIT (NPRO, LIBELLE, PRIX, QSTOCK)
S_COMMANDE(NCOM, NCLI, DATECOM)
S_DETAIL  (NCOM, NPRO, QCOM)
```

NCLI	NOM	ADRESSE	LOCALITE	CAT	COMPTE
B062	GOFFIN	72, r. de la Gare	Namur	B2	-3200
B112	HANSENNE	23, a. Dumont	Poitiers	C1	1250
B332	MONTI	112, r. Neuve	Geneve	B2	0
B512	GILLET	14, r. de l'Ete	Toulouse	B1	-8700
C003	AVRON	8, ch. de la Cure	Toulouse	B1	-1700
C123	MERCIER	25, r. Lemaitre	Namur	C1	-2300
C400	FERARD	65, r. du Tertre	Poitiers	B2	350
D063	MERCIER	201, bvd du Nord	Toulouse		-2250
F010	TOUSSAINT	5, r. Godefroid	Poitiers	C1	0
F011	PONCELET	17, Clos des Erables	Toulouse	B2	0
F400	JACOB	78, ch. du Moulin	Bruxelles	C2	0
K111	VANBIST	180, r. Florimont	Lille	B1	720
K729	NEUMAN	40, r. Bransart	Toulouse		0

29. En particulier, l'opérateur SQL de concaténation "||" s'écrit "&" en Access

L422	FRANCK	60, r. de Wepion	Namur	C1	0
S127	VANDERKA	3, av. des Roses	Namur	C1	-4580
S712	GUILLAUME	14a, ch. des Roses	Paris	B1	0

Figure A23.12.1 - Représentation des données sources de la table CLIENT

Chaque ligne de la table **S_<X>** représente une entité du type **<X>**. Elle comporte autant de colonnes que le type d'entités a d'attributs (A23.12.1). L'affichage du contenu de la table S_CLIENT s'obtient par la requête :

```
select NCLI, NOM, ADRESSE, LOCALITE, CAT, COMPTE
from S_CLIENT
```

A23.12.2Tables non structurées

La deuxième variante comporte toujours une table par type d'entités, soit ici quatre tables, mais chacune obéissant au même schéma :

```
M_CLIENT  (Entite, Attribut, Valeur)
M_PRODUIT (Entite, Attribut, Valeur)
M_COMMANDE(Entite, Attribut, Valeur)
M_DETAIL  (Entite, Attribut, Valeur)
```

Chaque ligne (e,a,v) de la table **M_<X>** représente la valeur d'un attribut d'une entité du type **<X>**. Plus précisément, elle indique que la valeur de l'attribut a de l'entité e est v. Une entité est représentée par une valeur d'un identifiant technique **eid**, une valeur arbitraire, non significative, unique dans sa table. Par simplicité, on a choisi la valeur de l'identifiant de l'entité dans la table **S_<X>** (pour DETAIL, on a construit les valeurs par concaténation des valeurs de NCOM et NPRO; p. ex. 30179-CS262), mais toute valeur, telle qu'un entier généré automatiquement, conviendrait. Chaque ligne de la table **S_<X>** est représentée dans la table **M_<X>** par autant de lignes que la table **S_<X>** a de colonnes (A23.12.2).

M_CLIENT		
Entite	**Attribut**	**Valeur**
B062	NCLI	B062
B062	NOM	GOFFIN
B062	ADRESSE	72, r. de la Gare
B062	LOCALITE	Namur
B062	CAT	B2
B062	COMPTE	-3200
B112	NCLI	B112
B112	NOM	HANSENNE
B112	ADRESSE	23, a. Dumont
B112	LOCALITE	Poitiers
B112	CAT	C1
B112	COMPTE	1250

...	...	...

Figure A23.12.2 - Représentation des données sous forme de tables à schéma unique

La construction des lignes de la table M_CLIENT à partir du contenu de la table S_CLIENT s'effectue par la suite de requêtes suivante :

```
insert into M_CLIENT ( Entite, Attribut, Valeur)
       select NCLI, 'NCLI', NCLI from S_CLIENT;
insert into M_CLIENT ( Entite, Attribut, Valeur)
       select NCLI, 'NOM', NOM from S_CLIENT;
insert into M_CLIENT ( Entite, Attribut, Valeur)
       select NCLI, 'ADRESSE', ADRESSE from S_CLIENT;
insert into M_CLIENT ( Entite, Attribut, Valeur)
       select NCLI, 'LOCALITE', LOCALITE from S_CLIENT;
insert into M_CLIENT ( Entite, Attribut, Valeur)
       select NCLI, 'CAT', CAT from S_CLIENT;
insert into M_CLIENT ( Entite, Attribut, Valeur)
       select NCLI, 'COMPTE', COMPTE from S_CLIENT;
```

A l'inverse, la reconstitution le la table source S_CLIENT s'obtient par la requête :

```
select M1.Valeur AS NCLI, M2.Valeur AS NOM,
       M3.Valeur AS ADRESSE,
       M4.Valeur AS LOCALITE, M5.Valeur AS CAT,
       val(M6.Valeur) AS COMPTE
from   M_CLIENT AS M1, M_CLIENT AS M2, M_CLIENT AS M3,
       M_CLIENT AS M4, M_CLIENT AS M5, M_CLIENT AS M6
where  M2.Entite = M1.Entite and M3.Entite = M1.Entite
and    M4.Entite = M1.Entite and M5.Entite = M1.Entite
and    M6.Entite = M1.Entite
and    M1.Attribut = 'NCLI'
and    M2.Attribut = 'NOM'
and    M3.Attribut = 'ADRESSE'
and    M4.Attribut = 'LOCALITE'
and    M5.Attribut = 'CAT'
and    M6.Attribut = 'COMPTE'
order by 1;
```

L'usage d'une simple jointure interne impose qu'il existe dans M_CLIENT une ligne pour chaque entité et pour chaque attribut, même si cette entité ne possède pas de valeur pour cet attribut. Si tel n'est pas le cas (pas de ligne si pas de valeur) on utilisera des jointures externes pour l'intégration des attribut facultatifs.

A23.12.3Table unique de quadruplets

La troisième variante est plus générale encore, puisqu'elle comprend une seule table par base de données. Elle dérive d'une union des tables de la représentation précédente. Chaque ligne indique la valeur d'un attribut d'une entité. Une quatrième colonne précise le type de l'entités.

```
MM_CLICOM(Type, Entité, Attribut, Valeur)
```

Chaque ligne (t,e,a,v) de la table **MM_<X>** représente la valeur d'un attribut d'une entité de la base de données **<X>**. Plus précisément, elle indique que la valeur de l'attribut a de l'entité e du type t est v (A23.12.3).

MM_CLICOM			
Type	**Entite**	**Attribut**	**Valeur**
CLIENT	B062	NCLI	B062
CLIENT	B062	NOM	GOFFIN
CLIENT	B062	ADRESSE	72, r. de la Gare
CLIENT	B062	LOCALITE	Namur
CLIENT	B062	CAT	B2
CLIENT	B062	COMPTE	-3200
CLIENT	B112	NCLI	B112
CLIENT	B112	NOM	HANSENNE
CLIENT	B112	ADRESSE	23, a. Dumont
CLIENT	B112	LOCALITE	Poitiers
CLIENT	B112	CAT	C1
CLIENT	B112	COMPTE	1250
. . .	. . .	. . .	. . .

Figure A23.12.3 - Représentation des données sous forme d'une seule table

La construction des lignes de la table M_CLICOM relatives au contenu de la table S_CLIENT s'effectue par la suite de requêtes suivante :

```
insert into MM_CLICOM (Type, Entite, Attribut, Valeur )
     select 'CLIENT', NCLI, 'NCLI', NCLI FROM S_CLIENT;
insert into MM_CLICOM (Type, Entite, Attribut, Valeur )
     select 'CLIENT', NCLI, 'NOM', NOM fromfrom S_CLIENT;
insert into MM_CLICOM (Type, Entite, Attribut, Valeur )
     select 'CLIENT', NCLI, 'ADRESSE', ADRESSE from S_CLIENT;
insert into MM_CLICOM (Type, Entite, Attribut, Valeur )
     select 'CLIENT', NCLI, 'LOCALITE', LOCALITE from S_CLIENT;
insert into MM_CLICOM (Type, Entite, Attribut, Valeur )
     select 'CLIENT', NCLI, 'CAT', CAT from S_CLIENT;
insert into MM_CLICOM (Type, Entite, Attribut, Valeur )
     select 'CLIENT', NCLI, 'COMPTE', COMPTE from S_CLIENT;
```

La reconstitution du contenu de la table S_CLIENT s'obtient par la requête :

```
select M1.Valeur AS NCLI, M2.Valeur AS NOM,
       M3.Valeur AS ADRESSE,
       M4.Valeur AS LOCALITE, M5.Valeur AS CAT,
       val(M6.Valeur) AS COMPTE
from   MM_CLICOM AS M1, MM_CLICOM AS M2, MM_CLICOM AS M3,
       MM_CLICOM AS M4, MM_CLICOM AS M5, MM_CLICOM AS M6
```

```
where  M2.Entite = M1.Entite and M3.Entite = M1.Entite
and    M4.Entite = M1.Entite and M5.Entite = M1.Entite
and    M6.Entite = M1.Entite
and    M1.Type = 'CLIENT' and M1.Attribut = 'NCLI'
and    M2.Type = 'CLIENT' and M2.Attribut = 'NOM'
and    M3.Type = 'CLIENT' and M3.Attribut = 'ADRESSE'
and    M4.Type = 'CLIENT' and M4.Attribut = 'LOCALITE'
and    M5.Type = 'CLIENT' and M5.Attribut = 'CAT'
and    M6.Type = 'CLIENT' and M6.Attribut = 'COMPTE'
order by 1;
```

Ici encore, les valeurs null sont représentées explicitement par une ligne dédiée à l'attribut de l'entité courante.

A23.12.4Table unique de triplets

Cette dernière représentation nous introduit dans un monde apparemment assez éloigné des bases de données mais proche de la technologie des ontologies.

```
MMM_CLICOM(Entité, Attribut, Valeur)
```

Toute information est spécifiée sous la forme d'un triplet (e, a, v) interprété comme suit : *la valeur de l'attribut a de l'entité e est v.* Elle pourrait sembler identique à celles que nous avons construites jusqu'ici, mais on remarquera que la notion de type d'entités a disparu. Celui-ci a en effet été réduit à une simple valeur de propriété. Toutes les données, qu'elles décrivent les entités ou les objets du schéma, sont regroupées dans une seule table **MMM_CLICOM** sous la forme de triplets. Les entités que représente le contenu des tables S_<X> reçoivent chacune un **eid absolu**. Deux entités de types différents reçoivent donc des eid distincts. Par simplicité, on a construit cet eid en reprenant la valeur de l'eid de la table **MM_CLICOM** préfixé du nom du type de l'entité (ex. CLIENT$B062) mais toute valeur, telle qu'un entier, serait parfaitement adéquat.

Dans cette représentation, il faut ajouter par rapport à la représentation précédente MM_CLICOM des types d'information spécifiant :

1. la structure des données (types d'entités et attributs)
2. le type de chaque entité.

Chacune de ces informations est représentée par un triplet (Entité, Attribut, Valeur). Les types d'entités et les attributs sont considérés comme des entités dont l'**eid** est constitué de leur nom. Dans la table **MMM_CLICOM**, cet eid peut apparaître comme **Entité** et comme **Valeur** (A23.12.4). Cette convention impose deux contraintes : les types d'entités ont des noms distincts (on représente donc une seule base de données) et les attributs ont des noms distincts (si deux types d'entités ont des attributs de même nom, ceux-ci sont un seul et même attribut). Ce principe peut être facilement étendu en rendant uniques les noms des attributs par concaténation de leur nom et de celui de leur type d'entités.

Pour la représentation des informations structurelles, on introduit deux nouvelles valeurs de la colonne **Attribut** :

- $is-of-type : spécifie que l'entité Entite est du type Valeur; exemples :
  ('CLIENT', '$is-of-type', 'ENTITY-TYPE'),
  ('COMPTE', '$is-of-type', 'ATTRIBUTE')
- $attribute-of : spécifie que l'entité Entité est un attribut de l'entité Valeur; exemple :
  ('LOCALITE', '$attribute-of', 'CLIENT')

MMM_CLICOM		
Entite	**Attribut**	**Valeur**
CLIENT	$is-of-type	ENTITY-TYPE
PRODUIT	$is-of-type	ENTITY-TYPE
COMMANDE	$is-of-type	ENTITY-TYPE
DETAIL	$is-of-type	ENTITY-TYPE
NCLI	$is-of-type	ATTRIBUTE
NOM	$is-of-type	ATTRIBUTE
ADRESSE	$is-of-type	ATTRIBUTE
LOCALITE	$is-of-type	ATTRIBUTE
COMPTE	$is-of-type	ATTRIBUTE
CAT	$is-of-type	ATTRIBUTE
NCLI	$attribute-of	CLIENT
NOM	$attribute-of	CLIENT
ADRESSE	$attribute-of	CLIENT
LOCALITE	$attribute-of	CLIENT
CAT	$attribute-of	CLIENT
COMPTE	$attribute-of	CLIENT
CLIENT$B062	$is-of-type	CLIENT
CLIENT$B112	$is-of-type	CLIENT
CLIENT$B062	NCLI	B062
CLIENT$B112	NCLI	B112
CLIENT$B062	NOM	GOFFIN
CLIENT$B112	NOM	HANSENNE
CLIENT$B062	ADRESSE	72, r. de la Gare
CLIENT$B112	ADRESSE	23, a. Dumont
CLIENT$B062	LOCALITE	Namur
CLIENT$B112	LOCALITE	Poitiers
CLIENT$B062	CAT	B2
CLIENT$B112	CAT	C1
CLIENT$B062	COMPTE	-3200
CLIENT$B112	COMPTE	1250

Figure A23.12.4 - Représentation des données sous forme de triplets uniformes

La conversion du contenu de la base de données CLICOM est obtenu par les requêtes suivantes :

```
-- métadonnées : types d'entités

insert into MMM_CLICOM (Entite, Attribut, Valeur )
values ( 'CLIENT', '$is-of-type', 'ENTITY-TYPE');
. . .

-- métadonnées : attributs

insert into MMM_CLICOM (Entite, Attribut, Valeur )
     values ( 'NCLI', '$is-of-type', 'ATTRIBUTE');
insert into MMM_CLICOM (Entite, Attribut, Valeur )
     values ( 'NCLI', '$attribute-of', 'CLIENT');
insert into MMM_CLICOM (Entite, Attribut, Valeur )
     values ( 'NOM', '$is-of-type', 'ATTRIBUTE');
insert into MMM_CLICOM (Entite, Attribut, Valeur )
     values ( 'NOM', '$attribute-of', 'CLIENT');
. . .

-- données : entités

insert into MMM_CLICOM (Entite, Attribut, Valeur )
     select 'CLIENT$'&NCLI,'$is-of-type','CLIENT' from S_CLIENT;

-- données : valeurs d'attributs

insert into MMM_CLICOM (Entite, Attribut, Valeur )
     select 'CLIENT$'&NCLI, 'NCLI', NCLI from S_CLIENT;
insert into MMM_CLICOM (Entite, Attribut, Valeur )
     select 'CLIENT$'&NCLI, 'NOM', NOM from S_CLIENT;
insert into MMM_CLICOM (Entite, Attribut, Valeur )
     select 'CLIENT$'&NCLI, 'ADRESSE', ADRESSE from S_CLIENT;
insert into MMM_CLICOM (Entite, Attribut, Valeur )
     select 'CLIENT$'&NCLI, 'LOCALITE', LOCALITE from S_CLIENT;
insert into MMM_CLICOM (Entite, Attribut, Valeur )
     select 'CLIENT$'&NCLI, 'CAT', CAT from S_CLIENT;
insert into MMM_CLICOM (Entite, Attribut, Valeur )
     select 'CLIENT$'&NCLI, 'COMPTE', COMPTE from S_CLIENT;
. . .
```

Le reconstitution du contenu de la table S_CLIENT s'obtient le plus simplement en deux temps. On construit d'abord la table MM_CLICOM par la requête suivante ou sous forme d'une vue SQL (les valeurs des eid sont différentes, ce qui est sans importance) :

```
insert into MM_CLICOM (Type, Entité, Attribut, Valeur)
select M1.Entite AS Type_Entite, M2.Entite AS Entite,
       M3.Entite AS Colonne, M4.Valeur
```

```
from   MMM_CLICOM AS M1, MMM_CLICOM AS M2,
       MMM_CLICOM AS M3, MMM_CLICOM AS M4
where  M1.Attribut = '$is-of-type'
and    M1.Valeur = 'ENTITY-TYPE'
and    M2.Attribut = '$is-of-type'   and M2.Valeur = M1.Entite
and    M3.Attribut = '$attribute-of' and M3.Valeur = M1.Entite
and    M4.Entite = M2.Entite and M4.Attribut = M3.Entite
order by M1.Entite, M2.Entite, M3.Entite;
```

On applique ensuite la requête de la section A23.12.3.

```
select M1.Valeur AS NCLI, M2.Valeur AS NOM,
       M3.Valeur AS ADRESSE,
       M4.Valeur AS LOCALITE, M5.Valeur AS CAT,
       val(M6.Valeur) AS COMPTE
from   MM_CLICOM AS M1, MM_CLICOM AS M2, MM_CLICOM AS M3,
       MM_CLICOM AS M4, MM_CLICOM AS M5, MM_CLICOM AS M6
where  M2.Entite = M1.Entite and M3.Entite = M1.Entite
and    M4.Entite = M1.Entite and M5.Entite = M1.Entite
and    M6.Entite = M1.Entite
```

A23.12.5Extensions

On observe que les descriptions du schéma (les métadonnées) n'ont pas été représentées de la même manière que les données, ceci afin de simplifier la manipulation de ces dernières. On pourrait normaliser leur représentation en considérant que les entités des types ENTITY-TYPE et ATTRIBUTE reçoivent elles-aussi un eid unique. Pour leur assigner un nom, on introduirait une nouvelle valeur d'attribut $nom. La déclaration

```
('CLIENT','$is-of-type','ENTITY-TYPE')
```

serait remplacée par les expressions plus régulières :

```
('abc45','$is-of-type','ENTITY-TYPE')
('abc45','$name','CLIENT')
```

La manipulation des données serait un peu plus complexe puisqu'elle nécessiterait une jointure supplémentaire. De même, en introduisant un nouvel objet BASE_DE_DONNEES, on pourrait se libérer de la contrainte de définition d'une table par base de données. Ces extensions sont laissées à l'initiative du lecteur.