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

Annexe **16**

Le modèle Entité-association étendu

Compléments

A16.1ATTRIBUTS ENTITÉ (ATTRIBUTS OBJET)

<complément de la section 16.7.4>

L'*attribut entité* (ou *attribut objet*) a pour domaine un type d'entités. Une valeur de cet attribut est une entité. Cette construction fait partie des modèles objet mais on peut en trouver des usages pertinents dans un schéma entité-association, où il peut remplacer de manière élégante certains types d'associations jugés *encombrants* (figure A16.1). On en trouvera plusieurs exemples dans les sections 16.9.10 et 16.14.

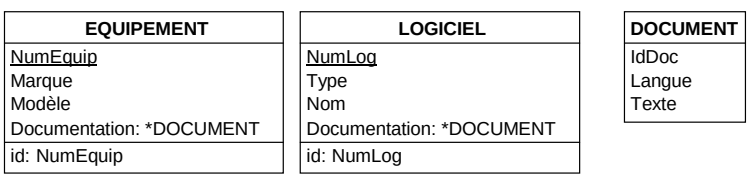


Figure A16.1 - Exemples d'attributs entité

A16.2TYPES D'ASSOCIATIONS DE TYPES D'ASSOCIATIONS

<complément de la section 16.8>

Certains auteurs[1] ont suggéré de généraliser les rôles à des types d'associations, de sorte qu'on puisse établir des associations entre une entité et une association ou entre deux associations (figure A16.2, montrant aussi un équivalent en structures classiques). Malgré son intérêt, cette construction n'a que rarement été introduite dans les modèles Entité-association en raison de sa complexité intrinsèque (par exemple, un type d'associations pourrait jouer l'un de ses propres rôles), des difficultés de compréhension qu'elle entraîne et des difficultés de traduction en structure de tables lisible. Elle n'a pas retenue dans le modèle utilisé dans cet ouvrage.

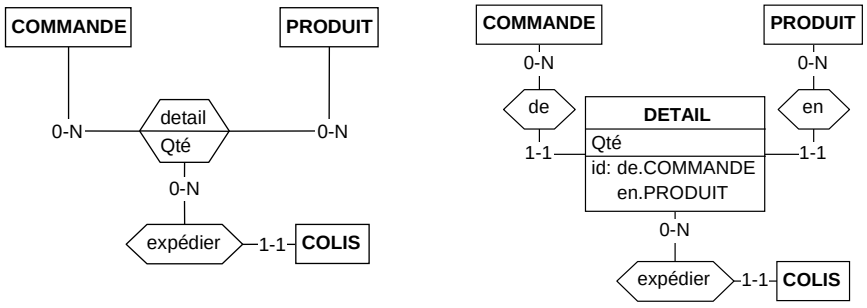


Figure A16.2 - Type d'associations de types d'associations (gauche) et une structure équivalente à droite

UML propose une construction hybride, la *classe-association*, qui est à la fois une association et une classe (section 17.10).

A16.3MUTATION D'UNE ENTITÉ

<complément de la section 16.9>

Une base de données n'est pas un instantané figé d'un domaine d'application. Bien au contraire, ce dernier évoluant constamment, le contenu de la base de données doit suivre le plus fidèlement possible cette évolution. Il est un mécanisme d'évolution très fréquent dans la réalité mais qui est souvent ignoré dans les bases de données : le **changement de type** d'une entité. Reprenons ci-après la figure 16.16 du chapitre 16. Nous y observons qu'un ouvrier est *affilié à une caisse d'assurance* et est *caractérisé par un matricule identifiant*, un *nom*, un *numéro d'affiliation* et un *régime salarial* ; en outre un *dossier personnel* peut lui être associé.

Si cet ouvrier décide d'ajouter une corde à son arc et de prendre à mi-temps le statut d'employé, l'entité qui le représente va acquérir des propriétés supplémentaires sous forme des attributs Service et Fonction.

1. Par exemple [Rochefeld, 1993].

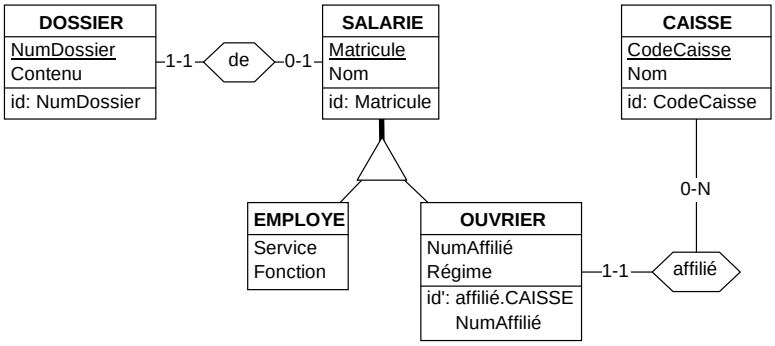


Figure A16.3 - Reprise de la figure 16.16

Plus tard, ce salarié peut décider de passer à 100% au statut d'employé. A partir de cet instant, il doit abandonner une partie de ses propriétés, celles définies par les attributs, rôles et contraintes propres de son ancien statut d'ouvrier.

Ce salarié peut enfin adopter un statut qui n'est pas répertorié dans le schéma. Dans ce cas, son entité appartient au type SALARIE mais à aucun de ses sous-types.

Les opérations d'entrée dans un type ou de sortie d'un type constitue une **mutation** de l'entité. Les contraintes structurelles qui s'imposent à l'entité en mutation sont entièrement définies par les caractéristiques des types en jeu.

A notre connaissance, les SGBD et les langages de programmation n'offrent pas de primitives correspondant à ces opérations de mutation. Leur traduction, qui sera souvent assez complexe, est malheureusement à charge du programmeur. On pourra s'inspirer de la référence ci-dessous.

Référence

Anne-France Brogneaux, Jean-Luc Hainaut. *Une approche complète de représentation des relations ISA en SQL2*, rapport final du projet Active Database, mars 2003, disponible à l'adresse https://projects.info.unamur.be/~dbm/mediawiki/index.php/LIBD:RAPPORTS

A16.4TYPES D'ENTITÉS ET CLASSES JAVA

<complément de la section 16.9>

Bien que le langage Java ne soit pertinent qu'aux niveaux logique et surtout physique, il est intéressant de comparer les hiérarchies de types d'entités avec celles des classes Java. Nous ferons deux observations.

La première concerne les contraintes existant entre une super-classe et ses sous-classes. La manière dont un objet est créé dans un programme Java induit le fait que cet objet n'est l'instance que d'une seule de ces sous-classes. Les sous-classes sont donc soumises à une contrainte de disjonction (**D**). D'autre part, lorsqu'une super-classe est déclarée abstraite, seules des instances de ses sous-classes (si elles-mêmes ne sont pas abstraites) peuvent être créées. Cette propriété revient à imposer une

contrainte de couverture totale (**T**) sur les sous-classes. Si au contraire une classe est concrète, alors il est possible d'en créer des instances. En résumé, le système de classes Java offre deux configurations sur les quatre décrites à la figure 16.15, soit **D** et **P**.

La deuxième observation concerne l'interprétation d'une relation *is-a* respectivement dans le domaine des bases de données et celui de la programmation. Dans le premier domaine, cette relation définit avant tout une propriété d'**inclusion des populations** de deux types d'entités, l'héritage n'étant qu'une conséquence logique de cette propriété. Dans le second domaine, cette relation définit un **mécanisme d'héritage** d'attributs et de méthodes. En général, la propriété d'inclusion des populations n'est pas pertinente en programmation (qu'est-ce que *l'ensemble des valeurs des variables* définies sur une classe ?). Coupler programmation OO et bases de données reste un défi que les SGBD *relationnels objet* ne relèvent qu'imparfaitement. Pour revenir à la section A16.9.6.5, on apprend qu'il n'existe pas d'interprétation unique du contenu de la table PERSONNE et de la table CLIENT. [Melton, 2003] propose trois réponses différentes à la question de la définition de leur contenu !

A16.5DU MAUVAIS USAGE DES RELATIONS IS-A

<complément de la section 16.9>

On trouvait il y quelques années, dans des manuels comportant une introduction à la programmation orientée objet, des schémas tels que celui de la figure A16.4 (gauche). Une telle illustration était destinée à faire comprendre au programmeur débutant l'essence de la programmation orientée objet.

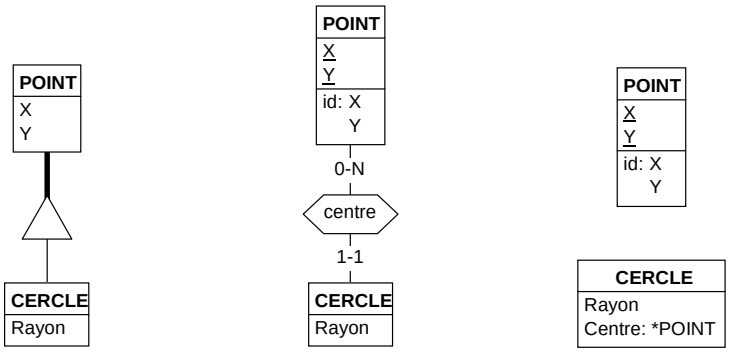


Figure A16.4 - Représentations erronée (gauche) et correctes (centre, droite) du fait qu'un cercle possède un centre repéré par ses coordonnées X et Y

On devine aisément le raisonnement des auteurs qui définissent CERCLE comme un sous-type de POINT :

> *je définis le cercle comme étant constitué d'un rayon et d'un centre, lui-même constitué des coordonnées X et Y ; je vais chercher ces dernières par héritage de la classe POINT qui existe déjà.*

Par cet exemple, on suggère que le cercle est une variante d'un point, ce qui exige une puissance d'imagination poétique que les concepteurs de bases de données consciencieux ont plutôt tendance à brider !

Techniquement les auteurs pensaient avoir obtenu ce qu'ils désiraient[2], mais conceptuellement leur proposition est une pure aberration. La solution correcte consiste à admettre qu'un cercle *n'est pas* un point mais *possède* un point (qui joue le rôle de centre). Il ne faut pas confondre *être* (schéma de gauche) et *avoir* (schéma central, par un type d'associations, et schéma de droite, par un attribut entité) ! On mesure mal, aujourd'hui encore, les dégâts cognitifs que cet exemple malheureux a provoqué dans l'esprit fragile des jeunes programmeurs de l'époque.

A16.6 RELATIONS IS-A ENTRE D'AUTRES COMPOSANTS DU MODÈLE

<complément de la section 16.9>

Pourquoi ne pourrait-on pas définir de relation *is-a* entre domaines, attributs et types d'associations ? L'idée est absolument légitime et a été proposée par plusieurs auteurs. La figure A16.5 montre un exemple représentatif dont l'interprétation est laissée aux bons soins du lecteur. De telles constructions n'ont cependant pas été retenues dans la plupart des modèles Entité-association, notamment en raison de la complexité de leur interprétation et des contraintes auxquelles elles sont soumises.

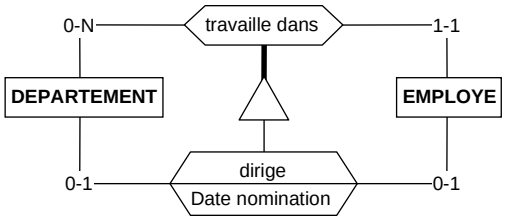


Figure A16.5 - Relation *is-a* entre deux types d'associations

A16.7 IDENTIFIANTS HYBRIDES GÉNÉRALISÉS

<complément de la section 16.10.1>

Un identifiant hybride mentionne un type d'entités voisin, connecté par un type d'associations. On pourrait admettre que ce voisin ne soit pas direct, mais connecté par deux, voire plusieurs types d'associations. Tel serait le cas du type SERVICE du schéma de la figure A16.5, dont les entités dépendant d'une même direction ont des valeurs de NomServ distinctes. Cette construction utile, qu'on rencontre notamment

2. Ce qui est d'ailleurs illusoire. Par exemple, si on admet qu'un point est identifié par ses coordonnées, il n'est pas possible selon ce schéma de définir deux cercles distincts de même centre !

dans les schémas XML (xsd), n'est pas malheureusement disponible dans les modèles Entité-associations.

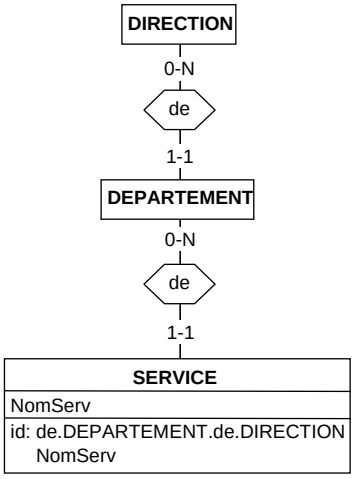


Figure A16.6 - Identifiant hybride généralisé

A16.8IDENTIFIANT CYCLIQUE D'UN TYPE D'ENTITÉS

<complément de la section 16.17>

Un identifiant hybride d'un type d'entités E est dit *cyclique* lorsqu'il comprend un rôle joué par E lui-même. On désire représenter un système de classification hiérarchique utilisé communément dans les bibliothèques[3]. Par exemple :

- la catégorie **3** représente les *Sciences sociales* ;
- la catégorie **39** représente *Ethnologie* ;
- la catégorie **398** représente *Folklore ;*
- la catégorie **3984** représente les *Le monde surnaturel ;*
- la catégorie **39844** représente les *Les dragons*.

Une catégorie est identifiée par un numéro au sein de sa catégorie supérieure, à l'exception des catégories principales, qui sont identifiées par leur numéro seul. Le schéma de la figure A16.7 (gauche) donne une première idée de structure.

3. Les codes illustrés sont extraits de la CDU (Classification Décimale Universelle).

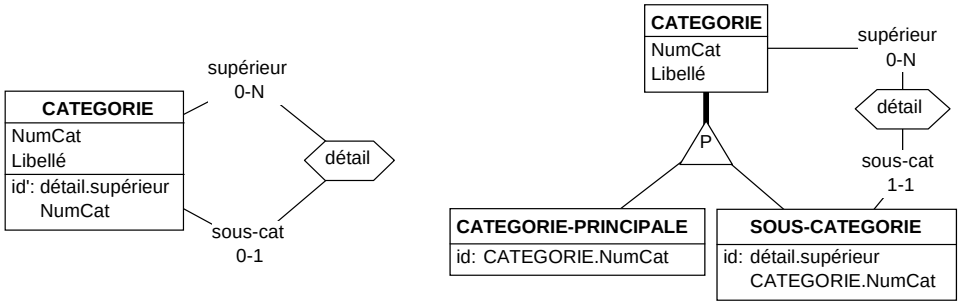


Figure A16.7 - Identifiant cyclique - Versions brute et affinée

L'attribut NumCat identifie la catégorie parmi toutes les sous-catégories de sa catégorie supérieure. L'identifiant ainsi défini n'est pas satisfaisant pour plusieurs raisons. D'une part, il comporte un composant obligatoire et un composant facultatif, ce qui constitue une construction problématique (il sera d'office secondaire). D'autre part, il convient pour les sous-catégories mais pas pour les catégories principales. En distinguant les deux types de catégories, on peut spécifier l'identifiant approprié de chacun (schéma de droite). Il est désormais possible de les déclarer primaires.

A16.9 IDENTIFIANT D'UN ATTRIBUT ET IDENTIFIANT DE SON TYPE D'ENTITÉS

<complément de la section 16.10.3>

Reprenons le schéma de droite de la figure 16.27, repris ci-après. Il indique que le composant NumCompte de Compte est unique parmi les clients. Il serait erroné, bien que tentant, de voir dans cet attribut Compte une représentation du concept de *compte personnel de client*, qui serait caractérisé par les propriétés *numéro de compte* (identifiant) et *montant du compte*. En particulier rien n'empêcherait qu'une même entité CLIENT possède deux valeurs de Compte dont les valeurs de NumCompte soient identiques et celles de Montant différentes.

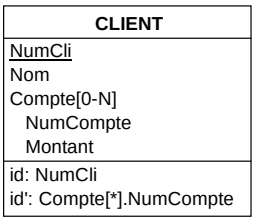


Figure A16.8 - Identifiant multivalué d'un type d'entités

Cas de figure troublant certes mais parfaitement conforme au schéma, lequel impose que deux entités CLIENT ne puissent partager la même valeur de NumCompte, mais

qui ne dit rien sur les valeurs au sein d'une même entité. Si en outre, on exige que les valeurs de NumCompte soient distinctes pour chaque entité CLIENT, alors il faut exprimer cette contrainte sous la forme d'un identifiant d'attribut (figure A16.9 A16.9, gauche). Cette forme n'est évidemment pas à recommander en raison de sa complexité. On la retrouvera par exemple dans des bases de données existantes dans un projet de rétro-ingénierie (chapitre A16.21). On lui préférera la variante de droite, qui lui est totalement équivalente mais qui est plus lisible.

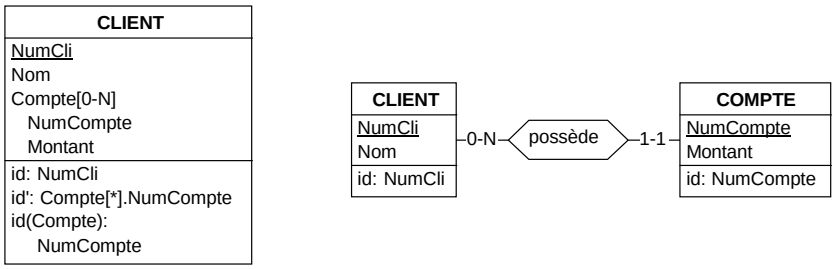


Figure A16.9 - Peut-on représenter un type d'entités muni d'un identifiant absolu (droite) sous la forme d'un attribut multivalué ? Oui, mais laborieusement (gauche) !

A16.10NOMS MULTIPLES, NOMS IDENTIQUES ET NOMS ABSENTS DES OBJETS D'UN SCHÉMA

Le principe d'unicité des noms peut être perçu comme trop contraignant, en particulier lors de la phase d'élaboration d'un schéma. L'imagination s'épuise rapidement devant la nécessité d'attribuer à 50 objets similaires des noms distincts pour la seule raison que le modèle l'exige. On voudrait parfois attribuer le même nom à plusieurs objets de même nature, notamment lorsqu'on désire seulement indiquer par ce nom la *nature* de ces objets et non un moyen de les distinguer les uns des autres.

Par exemple, le lien entre une commande et ses détails est de la même *nature* que celui qui relie une facture à ses détails. On dit en effet qu'*une commande* ***comporte*** *des détails de commande* et qu'*une facture* ***comporte*** *des détails de facture*. Dans certains cas, on aimerait même ne pas devoir nommer certains objets, soit temporairement, soit définitivement. Examinons deux techniques qui nous permettent de simplifier le dessin d'un schéma tout en respectant les règles d'unicité. Nous discuterons ensuite de la question des noms multiples d'un même objet.

a) Noms identiques (homonymes)

Lorsqu'un nom se termine par le symbole « | » (par exemple « passe| ») l'atelier DB-MAIN vérifie qu'il n'existe pas déjà un nom correspondant à la chaîne qui précède ce symbole (soit « passe »). Si ce nom existe déjà, l'atelier modifie le nouveau nom en lui ajoutant un suffixe qui le rend unique (par exemple « passe|_1 »). En outre, l'affichage ne montre que les caractères qui précèdent le

symbole « | » (soit ici « "passe" »). Le schéma de gauche de la figure A16.10.10 a été obtenu en attribuant au premier type d'associations le nom « "comporte|" » et au second le même nom « comporte| », lequel a été automatiquement transformé en « comporte|_1 ».

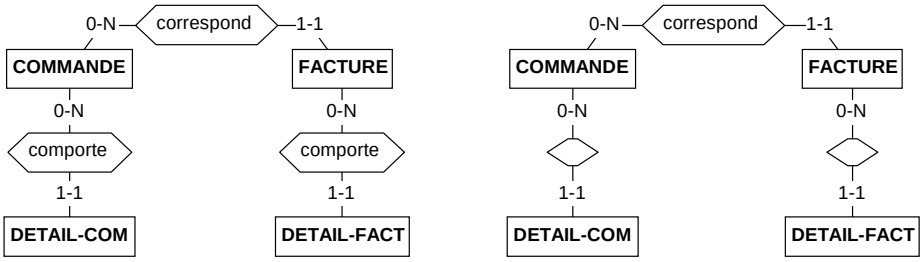


Figure 10.10 - Deux types d'associations homonymes et sans nom

b) Les objets sans nom

Il s'agit d'un cas particulier de noms identiques. Lorsque nous n'estimerons pas utile d'attribuer un nom à un objet, nous lui donnerons un nom vide « "|" », l'atelier DB-MAIN gérant alors automatiquement l'unicité (figure A16.10.10, droite).

c) Noms multiples d'un objet

Cf. Lexicon de DB-Main

A16.11CLASSES D'OBJETS GÉNÉRIQUES : LES TYPES D'ENTITÉS

<complément de la section 16.17>

Si les types d'associations génériques sont les plus connus et utilisés, on en trouve également parmi les autres classes d'objets d'un schéma. On examine ici deux catégories de types d'entités génériques : les *utilitaires* et les *dimensions*.

A16.11.1Types d'entités utilitaires

Un *type d'entité utilitaire* ne représente pas un concept spécifique du domaine d'application mais plutôt apporte une information complémentaire à un grand nombre d'autres types d'entités. La figure A16.11 montre trois approches de représentation du concept de *note* qui reprend des commentaires de nature quelconque qu'on désire associer aux entités de divers types, ici CONTRAT et AVENANT.

Le schéma de gauche introduit autant de *types d'associations* qu'il existe de types d'entités annotés ainsi qu'une contrainte d'existence volumineuse. Le schéma central crée un *surtype* pour tous ces types d'entités. Ces deux représentations sont correctes mais induisent une complexité graphique importante dès que le nombre de types d'entités concernés croît. La proposition de droite utilise des *attributs entité*,

particulièrement appropriés pour alléger le schéma.

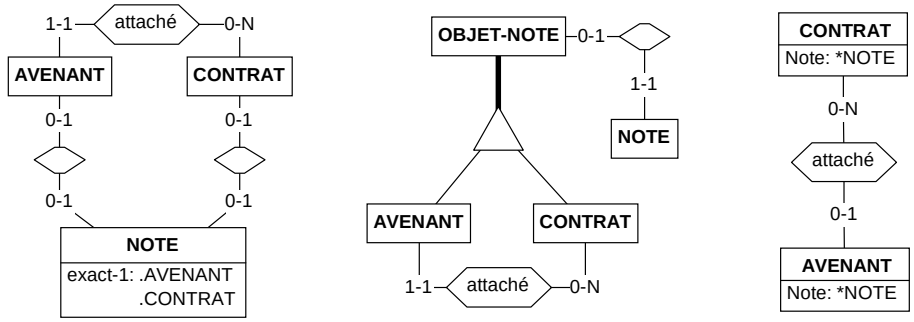


Figure A16.11 - Trois représentation du type d'entités utilitaire NOTE

A16.11.2Type d'entités dimension

Il s'agit d'un cas de figure assez proche du précédent à ceci près que le type d'entités représente un concept majeur, généralement transversal, du domaine d'application. Une dimension associée à un concept permet de décliner ce concept selon les différentes valeurs de la dimension. Les dimensions temporelles et spatiales sont bien connues, mais, d'une part, elle ne sont pas les seules, et, d'autre part, elle apparaissent souvent colorées par le domaine d'application.

Par exemple, *année* et *saison* sont des dimensions abstraites classiques, mais on les trouvera souvent sous forme de variantes telles que ANNEE-FISCALE, ANNEE-ACADEMIQUE ou SAISON-SPORTIVE. Dans un contexte international ou multi-lingue, la dimension LANGUE sera omniprésente. On citera encore à titre d'exemple les dimensions PAYS, DEPARTEMENT ou UNITE-MONETAIRE.

La figure A16.12 reprend un fragment du schéma d'une base de données de gestion d'une université. Les types d'entités COURS-TYPE et CLASSE-TYPE sont *dimensionnés* selon les années académiques ANNEE-ACA. Pour chaque entité COURS-TYPE, il existe une entité COURS par année académique. Le caractère dimensionnel de ANNEE-ACA se remarque par sa présence dans l'identifiant de COURS et CLASSE. On notera aussi que cette dimension est utilisée pour construire des *matérialisations* (section A16.16.14).

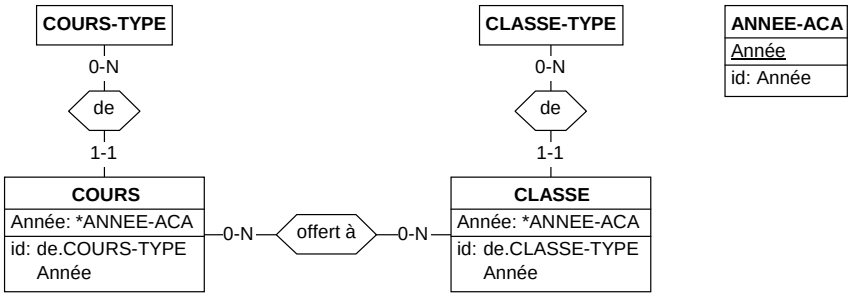


Figure A16.12 - Représentation de la dimension ANNEE-ACA(*DEMIQUE*)

A16.11.3 Autres classes génériques

Le concept de classe générique peut également concerner les attributs. Par exemple, « Date de création » ou « Propriétaire » pourraient être des attributs génériques pertinents pour de nombreux types d'entités d'un schéma, quel que soit le domaine d'application décrit.

A16.11.4 Objets du schéma ou du métaschéma ?

Bien qu'ils apparaissent dans des schémas liés chacun à un domaine d'application, les objets génériques en sont largement indépendants. D'une certaine manière, ils relèvent du modèle Entité-association. C'est l'approche qui est suivie par UML, où les associations de composition et d'agrégation font partir du metamodèle, au même titre que les classes d'objets, les attributs et les associations.

11.5 OPÉRATIONS ET MÉTHODES

<complément du chapitre 16>

Les modèles Entité-association récents ont intégré certains concepts des approches orientées objet. C'est ainsi qu'il est permis de spécifier le comportement dynamique d'un type d'entités, interprété comme une classe d'objets, sous la forme d'un ensemble de *méthodes*.

Une méthode est un service qui peut être demandé à toute entité d'un certain type. Elle se présente sous la forme d'une procédure dont l'exécution est déclenchée par un message envoyé à l'entité. Le schéma de la figure A16.13 spécifie que les actions dont les commandes peuvent faire l'objet sont : son enregistrement[4], la production d'une facture, son annulation, l'expédition des produits commandés (non illustrés dans le schéma) et son archivage.

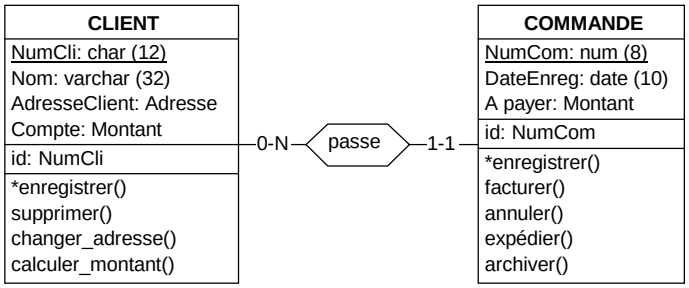


Figure A16.13 - Un type d'entités peut être doté de méthodes qui définissent son comportement

4. Le préfixe ∗ indique qu'il s'agit d'une méthode de classe et non d'instance : on demande à la classe COMMANDE d'enregistrer une nouvelle entité, mais on demande à une entité COMMANDE de se *facturer* ou de s'*archiver.*

A16.12SÉMANTIQUE DU MODÈLE ENTITÉ-ASSOCIATION

<complément de la section 16.17>

Nous avons évoqué à la section 15.3 deux acceptions différentes du concept de sémantique d'un modèle et de sémantique d'un schéma. Dans cette section, nous proposons une **sémantique interne** pour le modèle Entité-association basée sur le modèle relationnel du chapitre A16.3. L'élaboration d'une sémantique complète pour un modèle aussi riche serait une tâche complexe sans intérêt dans un ouvrage tel que celui-ci. Nous limiterons donc sa définition aux éléments importants du modèle, et en particulier aux contraintes d'intégrité. Nous établirons cette sémantique sous forme d'une correspondance entre les concepts du modèle Entité-association et une variante du modèle relationnel[5].

Le but de cette correspondance est de définir formellement et de préciser les concepts du modèle Entité-association afin qu'il ne subsiste aucune ambiguïté sur leur signification et leurs propriétés. Elle n'est en principe destinée ni à l'utilisateur ni même à l'analyste dans le cadre de ses activités professionnelles. Une des conséquences majeures de cette correspondance est que nous pourrons désormais appliquer tous les résultats développés pour le modèle relationnel au modèle Entité-association.

Nous étudierons successivement les types d'entités, les types d'associations, les attributs, les attributs et rôles obligatoires, les identifiants et les autres contraintes d'intégrité.

A16.12.1Types d'entités et relations *is-a*

La vision relationnelle d'un type d'entités est très simple : c'est celle d'un domaine particulier (dit *domaine-entité*) dont le nom est celui du type d'entités et dont chaque élément est une entité. On ne précise pas ce qu'on entend par une *entité*. On admet l'existence a priori de l'ensemble entités comprenant toutes les entités passées, présentes, futures ou simplement possibles. Ainsi, on définira le type d'entités SALARIE (figure 15.18) comme suit :

SALARIE $\subseteq$ entités

Afin d'éviter un excès de symboles mathématiques, on écrira plus simplement, pour définir les types d'entités SALARIE et CAISSE[6] :

SALARIE, CAISSE : entités

Les relations *is-a* s'interprètent par une simple propriété : la population du sous-type est un sous-ensemble de celle du surtype. OUVRIER et EMPLOYE seront donc définis par l'expression :

EMPLOYE, OUVRIER : SALARIE

5. Cette section s'inspire de [Hainaut, 1989] et [Hainaut, 2006].
6. Conformément à l'hypothèse formulée à la section 15.9.4 les domaines-entités définis comme des sous-ensembles directs de *entités* sont disjoints.

Nous reviendrons plus tard sur les propriétés des sous-types.

A16.12.2Types d'associations

La population d'un type d'associations binaire est un ensemble de couples d'entités et, en toute généralité, un type d'associations n-aire est un ensemble d'agrégats (ou n-uplet) de n entités. On retrouve donc une structure qui est celle des *relations* de la section A16.3.2.1, dont les domaines sont des populations de types d'entités. Chaque type d'associations de la figure 15.13 peut se définir sous la forme d'une relation dite *relation-association* (on ignore les cardinalités et les identifiants pour l'instant) :

spécifie(COMMANDE, PRODUIT, FOURNISSEUR)
offre(FOURNISSEUR, PRODUIT)

Lorsque le rôle porte un nom explicite (figure 15.9), celui-ci apparaît comme celui d'un attribut de la relation-association :

vendu(vendeur: CLIENT, acheteur: CLIENT, IMMEUBLE)

Un rôle polymorphique (figure 15.10) est défini sur l'union de domaines-entités :

possède(VEHICULE, propriétaire: PERSONNE $\cup$ SERVICE)

A16.12.3Attributs

Les attributs d'un type d'entités E sont décrits par une relation spéciale, dite *relation-entité*, qu'on nommera « descr-E » (*description de E*), composée d'une part d'un attribut-entité défini sur le domaine-entité E et d'autre part d'attributs représentant chacun un attribut du type d'entités[7]. Les attributs du type d'entités SALARIE seront définis comme suit (on ignore le type des attributs) :

descr-SALARIE(SALARIE, Matricule, Nom)

Chaque n-uplet est composé d'une entité SALARIE, d'une valeur de Matricule et d'une valeur de Nom. Il existe un seul n-uplet pour chaque entité SALARIE ; en outre, il n'existe qu'une seule entité SALARIE par valeur de Matricule. On a donc :

SALARIE $\longrightarrow$ Matricule, Nom
Matricule $\longrightarrow$ SALARIE

À partir de ces DF, on calcule aisément les deux identifiants de cette relation :

descr-SALARIE(<u>SALARIE</u>, <u>Matricule</u>, Nom)

Les attributs d'un type d'associations se définissent dans la relation-association elle-même. Les deux types d'associations de la figure 15.11 s'expriment comme suit :

réserve(DOCUMENT, EMPRUNTEUR, Date Réservation)
emprunte(EXEMPLAIRE, EMPRUNTEUR, PROJET, Date Emprunt, Date Retour)

7. Attention à ne pas confondre les *attributs Entité-association* que nous cherchons à représenter et les *attributs relationnels* par lesquels on les représente.

La caractère obligatoire ou facultatif des attributs sera étudié ci-après.

A16.12.4Attributs composés et multivalués

On notera d'abord que le modèle relationnel décrit au chapitre A16.3 se limite aux domaines simples et qu'il ignore donc les notions d'attribut composé et d'attribut multivalué. Leur prise en compte compliquerait considérablement le modèle relationnel et serait sans intérêt dans cet ouvrage. Nous nous contenterons donc de décrire de tels attributs sans développer des raisonnements et des processus aussi utiles que pour les attributs simples (opérations algébriques, DF, identifiants, normalisation, décomposition, etc.)

Un *attribut composé* peut être défini de deux manières : comme un attribut simple défini sur une relation existante ou comme un attribut dont on décrit localement le type de valeurs. Illustrons ces deux modes pour le type d'entités EMPRUNTEUR de la figure 15.4 (on ignore le type des attributs simples) :

Adresse-type(Rue, Ville)
descr-EMPRUNTEUR(EMPRUNTEUR, NumPers, Adresse: Adresse-type, ...)

descr-EMPRUNTEUR(EMPRUNTEUR, NumPers, Adresse(Rue, Ville), ...)

Un *attribut multivalué* est caractérisé par une propriété de cardinalité :

descr-EMPRUNTEUR(EMPRUNTEUR, NumPers, Adresse, Téléphone[1-5])

Un attribut *complexe* se définit par combinaison de ces caractéristiques (figure 15.4) :

descr-CLIENT(CLIENT, NCli, Contact[1-5] (Statut, Adresse(Rue, Ville), ...))

A16.12.5Attributs et rôles obligatoires

La relation descr-E, dont l'objectif est d'associer ses attributs au type d'entités E ne précise pas le caractère obligatoire des attributs. En effet, si une entité E n'apparaît pas dans descr-E, aucune valeur d'attribut ne lui est associée, de sorte que ses attributs sont tous facultatifs. Pour déclarer les attributs Matricule et Nom obligatoires pour SALARIE, il faut imposer une contrainte sur le domaine entité SALARIE qui précise que toutes les entités SALARIE apparaissent dans les n-uplets de descr-SALARIE :

descr-SALARIE[SALARIE] = SALARIE

Cette contrainte étant posée, un attribut obligatoire/facultatif d'un type d'entités s'exprime par un attribut obligatoire/facultatif de sa relation-entité. Le type d'entités ACCIDENT de la figure 15.1 se traduira par :

ACCIDENT : entités
descr-ACCIDENT(ACCIDENT, NumAcc, Montant[0-1])
descr-ACCIDENT[ACCIDENT] = ACCIDENT

Un rôle obligatoire se décrit de manière analogue (figure 15.1) :

appartient(VEHICULE, CLIENT)

appartient[VEHICULE] = VEHICULE

A16.12.6Identifiants

Les *identifiants locaux* sont traduits par des identifiants relationnels, tant pour les types d'entités que pour les types d'associations. Les exemples suivants sont tirés de schémas des chapitres A16.11 et 15 :

descr-CLIENT(CLIENT, NumClient, Nom, Adresse)
desc-VOL(VOL, Ligne, Date, NumAppareil)
desc-PROJET(PROJET, CodeProjet, Titre, Budget)
appartient(VEHICULE, CLIENT)
implique(VEHICULE, ACCIDENT)
offre(PRODUIT, FOURNISSEUR, Prix, Délai)
reserve(DOCUMENT, EMPRUNTEUR, Date Réservation)
emprunte(EMPRUNTEUR, PROJET, EXEMPLAIRE, Date Emprunt, Date Retour)

La représentation des *identifiants hybrides* est moins immédiate et nécessite un traitement préliminaire. Étudions l'identifiant de CONTRAT dans le schéma de la figure 15.1. Posons d'abord les définitions suivantes :

CONTRAT, CLIENT: entités
descr-CONTRAT(CONTRAT, NumCtr, Type, DateSign)
signe(CONTRAT, CLIENT)
descr-CONTRAT[CONTRAT] = CONTRAT
signe[CONTRAT] = CONTRAT

Les deux composants de l'identifiant hybride se trouvent dans des relations distinctes de sorte que leur expression est actuellement impossible. Observons d'abord que les deux contraintes d'égalité impliquent la suivante :

descr-CONTRAT[CONTRAT] = signe[CONTRAT]

En analysant attentivement ces définitions, on y retrouve un exemple du schéma de droite de l'équivalence suivante, extraite du chapitre A16.3 :

R(A,B,C,D)
A ⟶ B

⇔

R1(A,B)
R2(A,C,D)
R1[A] = R2[A]

Si on substitue signe à R1, descr-CONTRAT à R2 et à R et CONTRAT à A, on peut remplacer le fragment précédent par celui-ci, qui lui est équivalent (la DF étant induite par l'identifiant de desc-CONTRAT n'est pas notée) :

CONTRAT, CLIENT: entités
descr-CONTRAT(CONTRAT, CLIENT, NumCtr, Type, DateSign)
descr-CONTRAT[CONTRAT] = CONTRAT

On peut qualifier CLIENT d'*attribut-role*. Désormais, les deux composants de l'identifiant hybride de CONTRAT que nous cherchons à traduire se retrouvent dans la même relation. L'expression de cet identifiant est dès lors immédiate :

CONTRAT, CLIENT: entités
descr-CONTRAT(CONTRAT, CLIENT, NumCtr, Type, DateSign)
descr-CONTRAT[CONTRAT] = CONTRAT

En résumé, pour exprimer un identifiant hybride, on remplace la relation-entité (siège des éventuels composants attributs) et les relations-associations (sièges des composants rôles) impliquées dans sa définition par leur jointure selon le procédé précédent. L'identifiant hybride se traduit dans la relation étendue par un identifiant local. Étant donné les limites du modèle relationnel sur lequel nous nous appuyons, nous n'aborderons pas l'expression des autres formes d'identifiants.

A16.12.7 Autres contraintes

L'expression de la plupart des contraintes rencontrées dans ce chapitre ne pose pas de problèmes particuliers. Traitons quelques-unes d'entre elles.

a) Contrainte de cardinalité d'un rôle

Un fournisseur ne peut faire offre que de 0 à 10 produits (figure 15.13)[8] :

offre(FOURNISSEUR, PRODUIT, Prix, Délai)
$\forall$ f $\in$ FOURNISSEUR, $0 \leq |\text{offre(FOURNISSEUR=f)}| \leq 10$

On notera que cette contrainte ne peut se traduire par la cardinalité de l'attribut-entité de la relation. En effet, tout attribut-entité (exprimant un rôle) d'une relation-association est de cardinalité [1-1]. Notons encore que la contrainte [0-N] n'est pas exprimée, puisqu'elle correspond tout simplement à l'absence de contrainte !

b) Propriétés ensemblistes des sous-types

Exprimons les propriétés de *totalité* et de *disjonction* extraites du schéma 15.17 :

PERSONNE = CONTRIBUABLE $\cup$ MALADE

VEHICULE $\cap$ IMMEUBLE = $\varnothing$

c) Contraintes d'existence

Pour exprimer ces contraintes, nous introduisons la fonction logique non-null(A), utilisable dans l'opérateur algébrique de sélection, exprimant que l'attribut A (attribut normal ou attribut-rôle) du n-uplet courant possède une valeur. En considérant le schéma de la figure 15.39, on pourra écrire :

descr-EMPLOYE(EMPLOYE, Matricule, Nom,
SalaireHoraire[0-1], SalaireMensuel[0-1],
DateEngagement[0-1], PROJET[0-1])
descr-EMPLOYE(non-null(PROJET))[EMPLOYE]
= descr-EMPLOYE(non-null(DateEngagement))[EMPLOYE]
descr-EMPLOYE(non-null(SalaireHoraire))[EMPLOYE]

8. L'importance de cette contrainte suggère qu'on lui assigne une syntaxe spécifique, par exemple : card(offre.FOURNISSEUR) = [0-10]

∩ descr-EMPLOYE(non-null(SalaireMensuel))[EMPLOYE] = ∅

d) Dépendances fonctionnelles

Le schéma de la figure 15.40 se traduit (partiellement) comme suit :

ETUDIANT, MATIERE, PROFESSEUR: entités
descr-PROFESSEUR(PROFESSEUR, Matricule, Nom, Ecole, Directeur)
inscrit(ETUDIANT, MATIERE, PROFESSEUR)
descr-PROFESSEUR: Ecole ⟶ Directeur
inscrit: PROFESSEUR ⟶ MATIERE

Selon les principes de la normalisation étudiés au chapitre A16.3, il apparaît que les deux relations présentent des anomalies de redondance : aucun des deux déterminants n'est un identifiant de sa relation.

e) Contraintes cycliques

Traduisons les deux contraintes cycliques décrites à la section 15.12.3 :

départ[SECTION, STATION] ∩ arrivée[SECTION, STATION] = ∅
appartient[VEHICULE, CLIENT] = (couvre*signe)[VEHICULE, CLIENT]

A16.12.8Exemple complet

L'interprétation relationnelle du schéma de la figure 15.18 pourrait être la suivante :

DOSSIER, SALARIE, CAISSE : entités
EMPLOYE, OUVRIER : SALARIE
descr-DOSSIER(DOSSIER, NumDossier, Contenu)
descr-SALARIE(SALARIE, Matricule, Nom)
descr-EMPLOYE(EMPLOYE, Service, Fonction)
descr-OUVRIER(OUVRIER, NumAffilié, affilié: CAISSE, Régime)
descr-CAISSE(CAISSE, CodeCaisse, Nom)
de(DOSSIER, SALARIE)
descr-DOSSIER[DOSSIER] = DOSSIER
descr-SALARIE[SALARIE] = SALARIE
descr-EMPLOYE[EMPLOYE] = EMPLOYE
descr-OUVRIER[OUVRIER] = OUVRIER
descr-CAISSE[CAISSE] = CAISSE
de[DOSSIER] = DOSSIER

A16.12.9Application de la théorie relationnelle au modèle Entité-association

Nous avons vu que l'expression relationnelle du modèle Entité-association permettait de donner une signification mathématique précise et incontestable à chacun des objets et contraintes du modèle. On a également pu exprimer des contraintes et

propriétés de complexité quelconque[9]. Illustrons encore l'intérêt de cette sémantique interne par quatre applications particulièrement utiles.

f) Calcul des identifiants d'un type d'associations

Considérons le schéma de la figure 15.40. Le type d'associations inscrit peut se réécrire :

inscrit(ETUDIANT, MATIERE, PROFESSEUR)
inscrit: ETUDIANT, MATIERE ⟶ PROFESSEUR
inscrit: PROFESSEUR ⟶ MATIERE

On en déduit, selon la procédure Proc2, section A16.3.6.3, les deux identifiants {ETUDIANT, MATIERE} et {ETUDIANT, PROFESSEUR}.

g) Décomposition d'un type d'associations

Le schéma du type 1:N:N de la figure 15.34 possède un rôle de cardinalité [1-1]. Son expression relationnelle, selon les règles des identifiants implicites, est la suivante :

R(A, B, C)

L'existence des DF A ⟶ B, C nous autorise à décomposer R en R1 et R2 :

R1(A, B)
R2(A, C)
R1[A] = R2[A]

En réexprimant ce schéma dans le modèle Entité-associations, il vient (figure A16.14) :

Figure A16.14 - Normalisation d'un type d'associations

h) Normalisation d'un type d'associations

Le type d'associations inscrit du schéma de la figure 15.40 se traduit en une relation ternaire qui a fait l'objet de discussions au chapitre A16.3. Selon les identifiants qu'on a calculés en (a), cette relation n'est pas normalisée, mais elle est le siège d'un circuit de DF. Elle constitue ce qu'on a appelé un *noyau irréductible*, dont on sait (section A16.3.8.5) qu'il existe trois transformations possibles dénommées métaphoriquement *la peste*, *le choléra* et la *peste ET le choléra*. Chacun des schémas relationnels obtenus peut être réexprimé dans le modèle Entité-association, comme on l'illustrera par l'exercice 15.8.

9. Pour autant qu'elles ne fassent pas appel à la récursivité.

i) Héritage d'attributs d'un type d'entités

L'effet des mécanismes d'héritage peut se définir par des jointures. Considérons le schéma de la figure 15.18 dont l'expression relationnelle a été construite précédemment (A16.12.8). Les valeurs des attributs propres et hérités des entités EMPLOYE s'obtiennent par la jointure naturelle des deux relations-entité relatives à EMPLOYE et SALARIE :

descr-EMPLOYE(EMPLOYE)*descr-SALARIE(SALARIE)
[EMPLOYE,Matricule, Nom, Service, Fonction]

L'héritage ascendant s'obtiendrait par une jointure externe des deux relations-entité[10].

A16.12.10Interprétation relationnelle et traduction relationnelle

Il ne faut pas confondre l'*interprétation relationnelle* d'un schéma conceptuel que nous venons de définir avec sa *traduction* dans le modèle d'un SGBD relationnel.

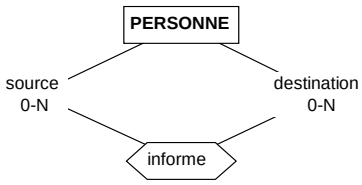
L'interprétation relationnelle a pour objectif de donner une sémantique au modèle, c'est-à-dire fournir une expression rigoureuse, non ambiguë des concepts d'un schéma. Elle n'a donc aucun objectif opérationnel.
La traduction d'un schéma Entité-association est le processus par lequel on produit une véritable base de données relationnelle (jusqu'au code SQL DDL) à partir d'un schéma conceptuel. Elle a été étudiée sommairement au chapitre A16.13 et nous y reviendrons en détail au chapitre A16.18 où elle sera l'objectif du processus de *conception logique*.

A16.13EXERCICES DU CHAPITRE 16

A16.1 On considère le schéma ci-dessous. Admettons à présent que, toutes choses restant égales par ailleurs, (1) un véhicule puisse être couvert par un nombre quelconque de contrats, (2) on désire connaître, chaque fois qu'un véhicule est impliqué dans un accident, le pourcentage de responsabilité qui lui a été attribué. Modifier le schéma en conséquence.

A16.2 On considère le type d'associations cyclique informe ci-dessous, qui représente les flux d'information entre des personnes. Si (p1, p2) ∈ PERSONNE, alors la personne p2 reçoit de l'information de la personne p1. On précise que ce type d'associations correspond à une relation transitive : si (p1, p2) et (p2, p3) existent, alors (p1, p3) existe également. Discutez de l'utilité de représenter par les instances toutes les associations ou seulement celles de la réduction transitive (qui ne sont pas obtenues par transitivité).

10. La jointure externe n'a pas été reprise dans le modèle relationnel que nous avons décrit en raison de l'introduction de valeurs null qui compliquent considérablement ce modèle. Cet opérateur a été décrit à la section A16.9.4.2 comme instruction SQL.

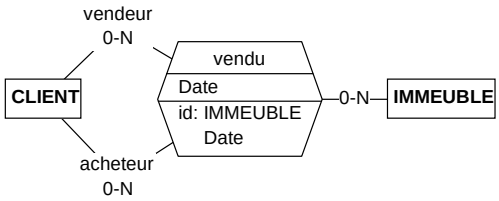


A16.3 Construire un tableau similaire à celui de la figure 15.34 pour les types d'associations binaires.

Le type d'associations cyclique vendu de la figure 15.9 n'est pas satisfaisant, car il impose certaines contraintes aux possibilités de vente. On suggère d'ajouter un attribut Date Vente. Compléter le schéma en introduisant cet attribut et en définissant l'identifiant qui semble le plus approprié.

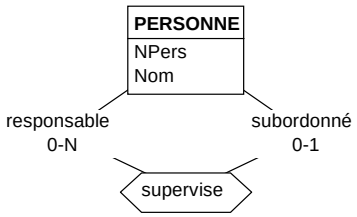
Solution

Ce schéma interdit à un vendeur de vendre plus d'une fois un même immeuble à un même acheteur. Situation rare mais possible ! En introduisant la *date de vente*, et en admettant qu'un immeuble ne puisse être vendu deux fois le même jour, il vient :



A16.4 Incroyable mais vrai !

On considère à nouveau le schéma cyclique bien connu repris ci-dessous.



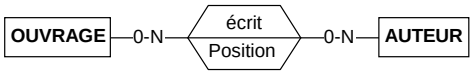
On observe que 80% des personnes ont un responsable et sont donc des subordonnés. Combien de subordonnés une personne supervise-t-elle en moyenne ?

Solution

L'énoncé nous apprend que $\mu_{subord} = 0{,}8$. Or, on sait (section 16.8.6) que $\mathbf{N}_{supervise} = \mathbf{N}_{pers} \times \mu_{resp} = \mathbf{N}_{pers} \times \mu_{subord}$. On en déduit l'égalité, peut-être un peu surprenante, $\mu_{resp} = \mu_{subord}$. Une personne supervise donc en moyenne 0,8 subordonnés.

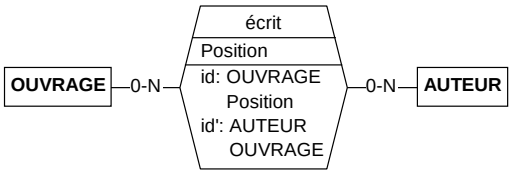
Cette question en entraîne une autre : combien **un responsable** supervise-t-il de subordonnés en moyenne ? Ce nombre est évidemment égal ou supérieur à 1 mais nous ne pouvons le calculer sans connaître une autre grandeur : la proportion de personnes qui sont responsables ou, autrement dit, la probabilité $\mathbf{p}_{resp}$ qu'une personne soit un responsable. Le nombre moyen d'associations supervise dans lesquelles une personne responsable apparaît est μ_{resp} / $\mathbf{p}_{resp}$. En supposant que 10% des personnes ont le statut de responsable, on déduit que celles-ci supervisent en moyenne une équipe de 0,8/0,1 = 8 personnes, valeur qui est sans doute plus naturelle !

A16.5 On considère le schéma ci-dessous, exprimant qu'un auteur apparaît dans une certaine position dans la liste des auteurs d'un ouvrage. Compléter ce schéma de manière à rendre explicite les deux règles métier : *un auteur n'apparaît qu'une seule fois pour un ouvrage*, et *il n'y a qu'un seul auteur à une position déterminée pour un ouvrage*. Ce schéma est-il normalisé ?



Solution

Cette question a été abordée dans l'exercice A16.3.5, où on a conclu que le schéma était normalisé.



A16.6 Représenter par un schéma un ensemble de personnes qui peuvent être unies par les liens du mariage. On représentera les faits suivants :

- il n'est pas interdit à une personne de se marier plusieurs fois,
- à une date donnée, une personne ne peut avoir qu'un seul conjoint,
- il n'est pas obligatoire d'être marié à tout instant,
- il n'est pas interdit à une personne d'épouser une même personne plus d'une fois.

A16.7 On considère le schéma de la figure 15.52. Combien ce schéma comporte-t-il de cycles ? Comment un type d'associations n-aire est-il pris en compte dans le repérage des cycles ? L'un de ces cycles fait-il l'objet de contraintes d'intégrité cycliques ?

A16.8 On considère les deux types d'associations r entre A et B et s entre B et C. On considère aussi le type d'associations rs défini comme la composition de r et s (figure A.A16.15) Calculer les cardinalités des rôles de rs à partir de celles des rôles de r et de s. On procédera en deux étapes :

1. on déterminera la classe fonctionnelle (1:N, 1:1, N:1 ou N:N) à partir de celles de r et de s,
2. on calculera les cardinalités [v-w] et [x-y] à partir de celles de r et de s.

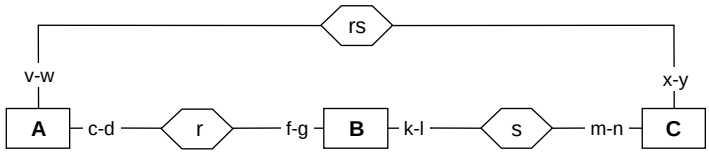
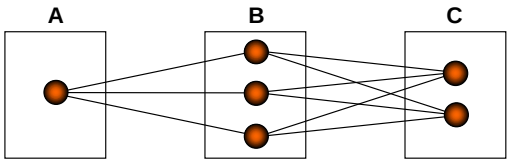


Figure A16.15 - Etude de la composition de deux types d'associations

Avant de crier prématurément victoire, on vérifiera que les formules obtenues, appliquées aux valeurs c = 3 et k = 2, décrivent correctement le schéma d'instances ci-dessous :



Solution

On ne traite ici que la question 2. La question 1, assez simple, est reportée au chapitre 11.

Première approche

Une entité A quelconque apparaît dans au plus **d** associations r et est donc associée à **d** entités B au plus. Chacune de celles-ci apparaît dans au plus **l** associations s et donc est associée à **l** entités C au plus. On suggère donc **w** = **d** x **l**, et, par symétrie, **y** = **m** x **f**.

Pour les cardinalités minimales **v** et **x**, on pourrait a priori appliquer un raisonnement similaire : une entité A apparaît dans au moins **c** associations r et est donc associée à au moins **c** entités B. A chacune de celles-ci correspondent au moins **k** entités de C. On aurait donc **v** = **c** x **k**, et, par symétrie, **x** = **n** x **g**.

Malheureusement, ce dernier raisonnement est trop simpliste, comme l'illustre le schéma d'instances de l'énoncé, qui représente des instances valides du schéma pour **c** = **3** et **k** = **2**. On y observe qu'il existe bien **c** x **k** = **6** associations s, mais qu'une entité A n'est associée via rs qu'à **2** entités C, et non **6** au moins comme le voudrait la règle ci-dessus.

Analyse

Il apparaît en effet que deux entités B distinctes peuvent être associées à une même entité C. Dans un tel cas, le nombre d'entités C associées à une entité A pourra être inférieur à **c** x **k**. La valeur **c** x **k** représente donc la valeur maximale de **v**, mais certaines configurations admettent des valeurs inférieures. Il n'est pas difficile de déterminer ce qui caractérise ces situations :

pour qu'une même entité C puisse être partagée par plusieurs entités B, il faut que celle-ci puisse apparaître dans plusieurs associations s, et donc il faut que **m** > **1**.

La règle **v** = **c** x **k** n'est donc correcte que lorsque **c = 0** ou **k = 0** (auxquels cas **v = 0**), ou lorsque **m** = **1**. Elle doit être affinée lorsque **c** et **k > 0** et **m** > **1**, ce que nous allons faire.

Etudions d'abord jusqu'où nous pouvons légalement réduire le nombre d'entités C. Nous savons qu'à toute entité A correspondent (au moins) **c** x **k** associations s. Or, une entité C ne peut apparaître dans plus de **m** associations s. Donc ces **c** x **k** associations s exigent au moins $\lceil$**c** x **k/m**$\rceil$ entités C distinctes. On a donc **v** ≥ $\lceil$**c** x **k/m**$\rceil$.

Observons ensuite que cette réduction est limitée par une autre contrainte : puisqu'une entité B doit apparaître dans au moins **k** associations s, il faut pour construire celles-ci au moins **k** entités C distinctes, le nombre total d'entités C ne peut être inférieur à **k**. On a donc aussi **v** ≥ **k**.

On en conclut que **v** ≥ max($\lceil$**c** x **k/m**$\rceil$**, k)** lorsque **c > 0** et **k > 0** et **m** > **1**.

Nous disposons à présent de deux valeurs pour **v** : une valeur maximale (**c** x **k**), valable lorsque **c** = 0 ou **k** = 0 ou **m** =1 et une valeur minimale (max($\lceil$**c** x **k/m**$\rceil$, **k**)) valable dans les autres cas. On unifie ces deux cas par la règle

v = min(**c** x **k**, max($\lceil$**c** x **k/m**$\rceil$, **k**)).

Conclusion

On peut donc écrire :

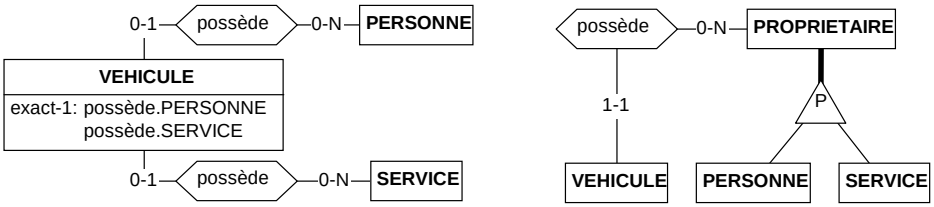
v = min(**c** x **k**, max($\lceil$**c** x **k/m**$\rceil$, **k**))
w = **d** x **l**
x = min(**n** x **g**, max($\lceil$**n** x **g/d**$\rceil$, **g**))
y = **m** x **f**

Remarque

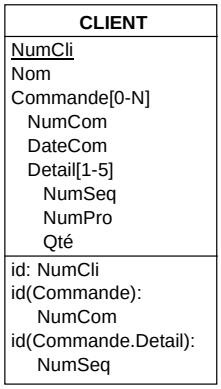
Afin que les tailles des populations des types d'entités n'imposent pas de contraintes sur les cardinalités, on suppose que ces tailles sont très grandes vis-à-vis de d, g, l, n.

A16.9 Proposer deux formes équivalentes du type d'associations multi-type d'entités de la figure 15.10. Voir aussi , sur le même thème, l'exercice 17.21.

Solution

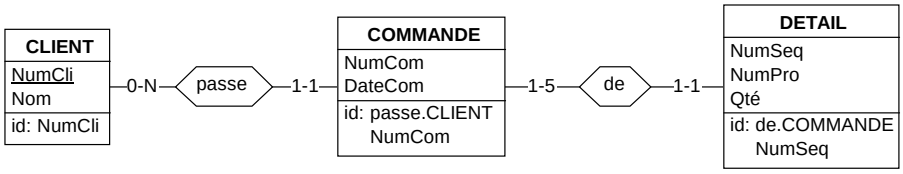


A16.10 Un attribut de structure trop complexe peut être l'indice d'une mauvaise modélisation. Montrer que l'exemple ci-contre en est une illustration. Proposer une variante équivalente plus appropriée.



Solution

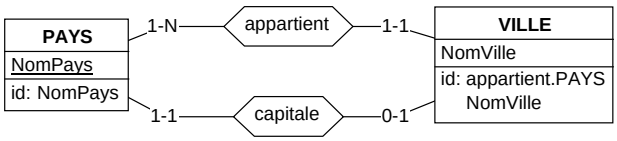
L'attribut complexe Commande est extrait sous la forme du type d'entités COMMANDE. Son attribut complexe Détail de ce dernier est traité de la même manière. On obtient dès lors le schéma ci-dessous :



A16.11 On considère le schéma de droite de la figure 15.41, représentant des nomenclatures de produits. On a convenu que le graphe des instances ne comportait pas de circuits (15.12.4). Cette propriété garantit par exemple que des procédures récursives travaillant sur les instances ne boucleront pas.

Admettons maintenant que certains produits finis sont réinjectés dans le processus de fabrication, comme c'est le cas dans la métallurgie et dans l'industrie verrière, où les chutes et les pièces non conformes sont recyclées, de sorte qu'un produit fini est aussi une matière première. Modifiez le schéma de manière (1) à tenir compte de l'existence de tels circuits et (2) à conserver la propriété d'acyclicité initiale.

A16.12Compléter le schéma ci-dessous des contraintes qui paraissent évidentes.



Solution

Si on admet que la capitale d'un pays appartient à ce dernier, alors il faut imposer la contrainte capitale ⊆ appartient.

A16.13Compléter le schéma de la figure A16.16 en calculant les grandeurs inconnues, notées "?".

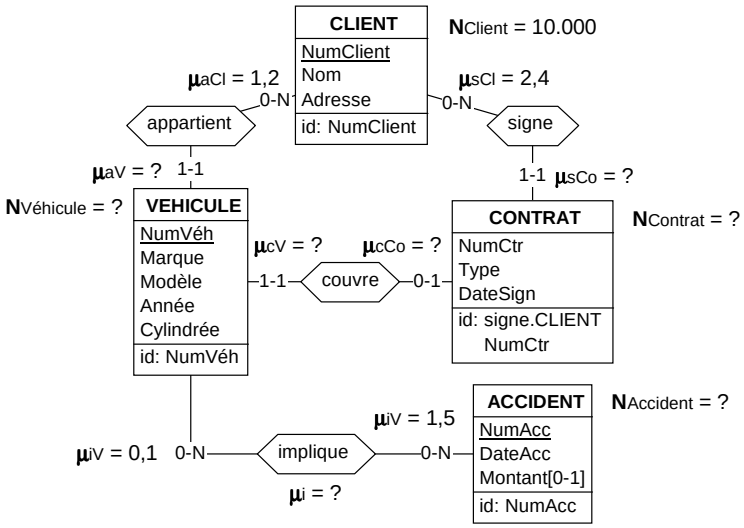


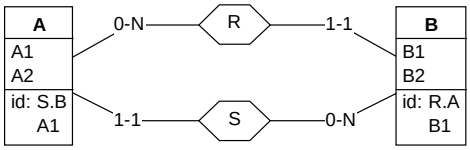
Figure A16.16 - Calcul des tailles des populations

Solution

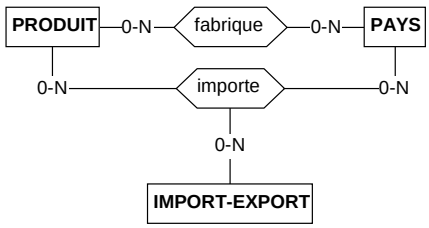
En exploitant les relations de la section A16.16.8.6, il vient :

appartient :	$\mu_{aV} = 1$	ACCIDENT :	$N_{Accident} = 800$
signe :	$\mu_{sCo} = 1$	couvre :	$\mu_{cV} = 1$
VEHICULE :	$N_{Véhicule} = 12\,000$	couvre :	$\mu_{cCo} = 0{,}5$
CONTRAT :	$N_{Contrat} = 24\,000$	implique :	$\mu_{i} = 1\,200$

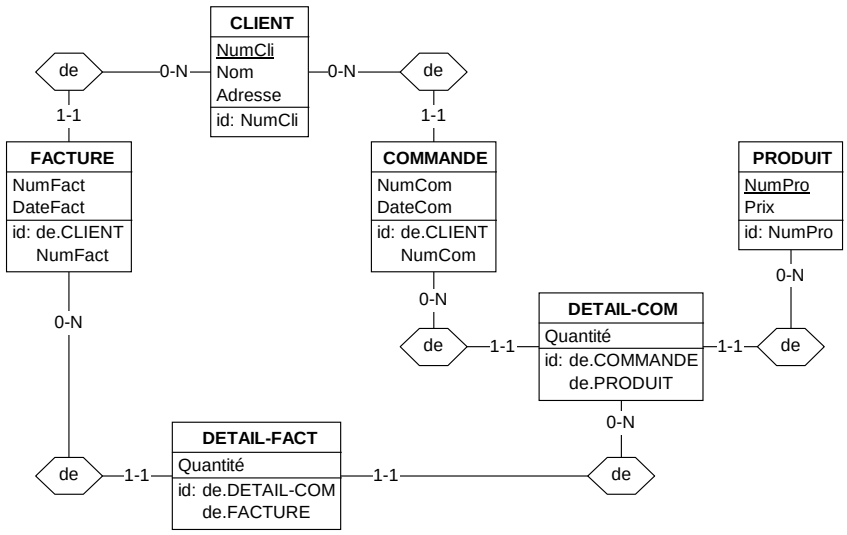
A16.14Analyser le schéma ci-dessous.



A16.15Ajouter au schéma ci-dessous la contraints selon laquelle un pays n'importe pas un produit qu'il fabrique.



A16.16Indiquer pour chaque type d'entités du schéma ci-dessous les attributs pour lesquels on doit fournir une valeur de manière à identifier une entité de ce type. Par exemple, pour identifier une entité CLIENT, il faut fournir une valeur de CLIENT.NumCli, tandis que pour une entité FACTURE, il faut fournir une valeur de de.CLIENT.NumCli et une valeur de FACTURE.NumFact[11].



Solution

11. Attention, ceci ne veut pas dire que le groupe {NumCli, NumFact} constitue un identifiant de FACTURE.

Pour identifier ...	**il faut connaître les valeurs de ...**
CLIENT	CLIENT.NumCli
FACTURE	de.CLIENT.NumCli, FACTURE.NumFact
COMMANDE	de.CLIENT.NumCli, COMMANDE.NumCom
PRODUIT	PRODUIT.NumPro
DETAIL-COM	de.COMMANDE.de.CLIENT.NumCli, de.COMMANDE.NumCom, de.PRODUIT.NumPro
DETAIL-FACT	de.FACTURE.de.CLIENT.NumCli, de.FACTURE.NumFact, de.DETAIL-COM.de.COMMANDE.de.CLIENT.NumCli, de.DETAIL-COM.de.COMMANDE.NumCom, de.DETAIL-COM.de.PRODUIT.NumPro

L'identification de DETAIL-FACT nécessite la connaissance de deux valeurs de NumCli.

A16.17 Le schéma de la question A16.16 est-il le siège d'une contrainte cyclique ? Si oui, en écrire l'expression.

A16.18 Proposer un méta-schéma pour les modèles SQL2 et SQL3.

A16.19 Proposer un méta-schéma pour le modèle relationnel (chapitre 3).

A16.20 Proposer un méta-schéma pour le modèle Entité-association étendu.

A16.21 Imaginer un schéma équivalent à celui de la figure 16.35 (droite) mais utilisant un type d'associations de composition.

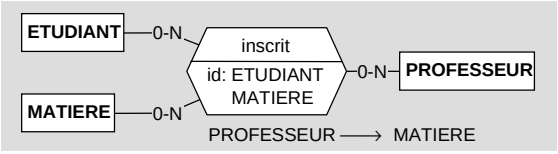
A16.22 Le schéma de droite de la figure 16.39 décrit partiellement la composition d'un véhicule. Son interprétation est cependant loin d'être évidente. Le lecteur est invité à examiner les questions suivantes, et à modifier ou compléter le schéma si nécessaire.

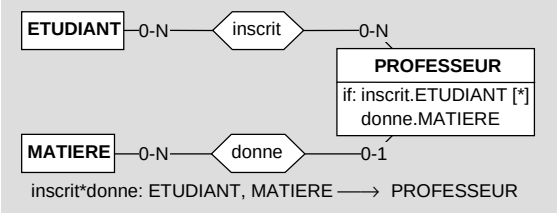
- Que représente une entité MOTEUR ? Un type de moteur ou un moteur bien précis ?
- Une entité MOTEUR existe-elle indépendamment de l'entité VEHICULE dans la composition de laquelle elle entre ? Un moteur peut-il changer de véhicule ? Un véhicule peut-il changer de moteur ?
- Le type d'entités ROUE représente-t-il les *types de roue* disponibles ou bien toutes les *roues* fabriquées ? Comment modifier le schéma pour tenir compte de l'autre interprétation ?

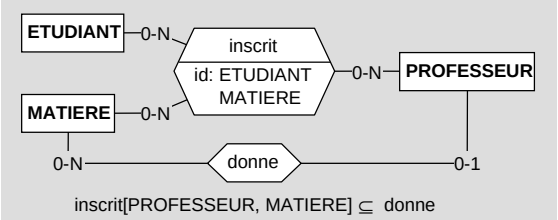
A16.23On a montré dans la section 15.18.9, point (c), que le type d'associations inscrit de la figure 15.40 n'était pas normalisé et qu'il pouvait se transformer en trois schémas équivalents. Dessiner ces schémas dans le modèle Entité-association en indiquant soigneusement toutes les contraintes d'intégrité.

Solution

En s'inspirant des solutions obtenues à la fin de la section A16.3.8.5, on obtient :







A16.24Définir la sémantique relationnelle du schéma 11.27 (*Voyages en train*).

A16.25Définir la sémantique relationnelle du schéma 15.17.

A16.26Définir la sémantique relationnelle du schéma 15.52.

A16.27On considère un type d'entités FEMME doté des attributs Nom et Prénom. Compléter ce schéma en y incluant la représentation des deux propriétés suivantes :

a) une femme peut être la mère d'autres femmes,

b) une mère donne à ses filles des prénoms distincts.

Solution

Il s'agit d'un exemple d'identifiant cyclique (voir section 15.10.2).

A16.28Représenter la structure et la croissance de certains arbres fruitiers (en l'occurrence des pommiers) qui obéissent aux lois suivantes. 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[12].

Solution

Il s'agit d'un exemple d'identifiant cyclique (voir section 15.10.2).

A16.29Le méta-schéma de la figure 16.43 représente des identifiants qui traduisent l'unicité des noms des objets d'un schéma. Cependant, la contrainte qui veut que les *noms des rôles d'un type d'associations soient distincts* n'est pas traduite correctement si on inclut dans ces noms les noms par défaut. Corriger ce schéma pour tenir compte de ces derniers.

A16.30Considérons le tableau des contraintes d'existence de la section 16.11.3. Le lecteur quelque peu familier avec la logique aura reconnu dans ce tableau une table de vérité définissant complètement cinq fonctions logiques à deux variables (`A` et `B`). Sachant qu'il existe 16 fonctions logiques à deux variables, on observe que ce tableau n'est pas complet. On peut représenter chaque fonction par les valeurs présentes dans sa colonne lues de haut en bas. Ainsi, `coex` = [`1001`] et `excl` = [`1110`]. Etudier les autres fonctions : sont-elles utiles ? Existe-t-il des fonctions primitives, à partir desquelles il est possible de construire les autres ?

Solution

Le tableau ci-dessous complète celui de la section 16.11.3 en représentant les 16 fonctions. On attribue, en entête de chaque colonne, un numéro de `0` à `15` à chaque fonction. Ainsi, `coex` = [`1001`] reçoit le numéro `9` et `excl` = [`1110`] le numéro `14`.

A	B	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	**0**	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	**1**	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	**0**	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	**1**	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		-	-	-		-				-							-

On peut faire quelques observations intéressantes.

12. Problème aimablement communiqué par J-J. Claustriaux, professeur à la Faculté Agronomique de Gembloux.

- Certaines fonctions n'ont aucune utilité. Par exemple, la fonction 0 ([0000]) rend le parent *non satisfiable* et la fonction 15 ([1111]) n'impose aucune contrainte. Nous pouvons les ignorer.
- De même, les quatre fonctions ne comportant qu'un seul signe 1 sont sans intérêt, puisqu'on peut toujours leur substituer une modification du schéma : supprimer A et/ou B ou rendre A et/ou B obligatoire. On ignore donc aussi les fonctions 1, 2, 4 et 8.
- Restent ainsi les dix fonctions utiles 3, 5, 6, 7, 9, 10, 11, 12, 13 et 14. Certaines comporte trois signes 1 (7, 11, 13, 14) et les autres deux signes 1 (3, 5, 6, 9, 10, 12).
- On observe que toutes les fonctions à deux signes 1 peuvent être obtenues par une conjonction de deux fonctions à trois signes 1. Par exemple, [1010] = [1011] $\wedge$ [1110].
- On en conclut que les quatre fonctions (7, 11, 13, 14) sont primitives et permettent de reconstruire toutes les fonctions utiles. Elles correspondent aux contraintes "at-lst-1: A, B", "if: B, A", "if: A, B" et "excl: A, B".
- Notre modèle Entité-association comporte donc trois contraintes d'existence primitives **at-lst-1**, **if** et **excl**.

A16.31 Etablir les tables de vérité des contraintes d'existence portant respectivement sur 3 et 4 composants facultatifs.

Solution

Cette question sera résolue pour les expressions à 2 et 3 composants à la section 20.5. On y montre qu'il existe une loi de formation qui permet d'extrapoler l'expression des contraintes de plus de 3 composants.

A16.32 On considère les contraintes excl: A, B, C et coex: A, B, C. Exprimer ces contraintes en utilisant des contraintes à deux composants.

A16.33 En examinant la table de vérité évoquée à l'exercice A16.30 et en considérant la fonction logique existe(x.E), qui renvoie vrai si le composant E de l'objet x existe et faux sinon, établir la contrainte, ou la conjonction de contraintes, équivalente à chacune des expressions suivantes :

équivalence : existe(x.A) = existe(x.B)

disjonction exclusive : existe(x.A) $\oplus$ existe(x.B)

implication : existe(x.A) $\Rightarrow$ existe(x.B)

disjonction : existe(x.A) $\vee$ existe(x.B)

conjonction : existe(x.A) $\wedge$ existe(x.B)

A16.34 Comment exprimer à l'aide des contraintes d'existence la propriété qui dit que *si A existe, alors B doit être absent*.

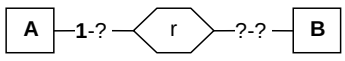
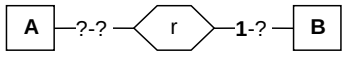
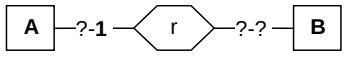
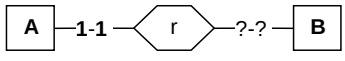
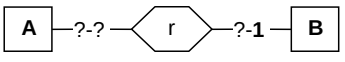
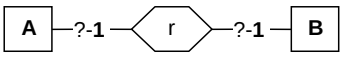
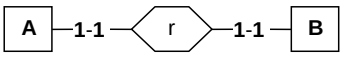
Solution

Si A est absent, la propriété est vraie. Si A est présent et B absent la propriété est vraie. Si A est présent et B l'est aussi la propriété est fausse. Cette propriété correspond donc à la fonction [1110], qui est l'expression de **excl: A, B**.

A16.35Un type d'associations binaire R entre A et B a pour équivalent mathématique une relation binaire R(A,B). Les mathématiciens ont identifié des classes remarquables de relations binaires : *partout définie*, *surjection*, *fonction*, application, *injection*, *bijection*. Quelles sont les propriétés du type d'associations R qui correspondrait à chacune de ces classes ?

Solution

En considérant R de A vers B, on peut définir les équivalences suivantes[13] :

classe	équivalent algébrique	TA
partout définie	R[A] = A	
surjection	R[B] = B	
fonction	R(A,B)	
application	R(A,B); R[A] = A	
injection	R(A,B)	
bijection (faible)	R(A,B)	
bijection (forte)	R(A,B); R[A]=A; R[B]=B	

13. Nous avons qualifié de *faible* le concept de bijection adaptée au domaine des bases de données (relation fonctionnelle injective). Les mathématiciens considèrent habituellement une définition *forte* : application (fonction partout définie) surjective et injective.

A16.36En utilisant les résultats de la question A16.30, simplifier chacun des schémas suivants.