Conception de bases de données avec UML

GILLES ROY

Presses de l'Université du Québec

Conception de bases de données avec UML

PRESSES DE L'UNIVERSITÉ DU QUÉBEC
Le Delta I, 2875, boulevard Laurier, bureau 450
Québec (Québec) G1V 2M2
Téléphone : 418-657-4399 ▪ Télécopieur : 418-657-2096
Courriel : puq@puq.ca ▪ Internet : www.puq.ca

Diffusion / Distribution :

CANADA et autres pays
PROLOGUE INC.
1650, boulevard Lionel-Bertrand
Boisbriand (Québec) J7H 1N7
Téléphone : 450-434-0306 / 1 800 363-2864

FRANCE
AFPU-DIFFUSION
SODIS

BELGIQUE
PATRIMOINE SPRL
168, rue du Noyer
1030 Bruxelles
Belgique

SUISSE
SERVIDIS SA
Chemin des Chalets
1279 Chavannes-de-Bogis
Suisse

Conception de bases de données avec UML

GILLES ROY

2009

Presses de l'Université du Québec
Le Delta I, 2875, boul. Laurier, bur. 450
Québec (Québec) Canada G1V 2M2

Catalogage avant publication de Bibliothèque
et Archives nationales du Québec et Bibliothèque et Archives Canada

Roy, Gilles, 1951-

Conception de bases de données avec UML

Comprend des réf. bibliogr. et un index.

ISBN 978-2-7605-1500-0

1. Bases de données - Conception. 2. UML (Informatique). 3. Modèles entité-association. 4. Bases de données relationnelles. 5. Structures de données (Informatique). I. Titre.

QA76.9.D26R69 2007 005.74 C2007-940915-6

Nous reconnaissons l'aide financière du gouvernement du Canada par l'entremise du Programme d'aide au développement de l'industrie de l'édition (PADIE) pour nos activités d'édition.

La publication de cet ouvrage a été rendue possible grâce à l'aide financière de la Société de développement des entreprises culturelles (SODEC).

Mise en pages : INFOSCAN COLLETTE-QUÉBEC

Couverture : RICHARD HODGSON

1 2 3 4 5 6 7 8 9 PUQ 2009 9 8 7 6 5 4 3 **2** 1

Dépôt légal – 3e trimestre 2007
Bibliothèque et Archives nationales du Québec / Bibliothèque et Archives Canada
Imprimé au Canada

Table des matières

Remerciements

Merci tout d'abord à mes amis et collègues de l'Université du Québec à Rimouski (UQAR), Campus de Lévis, qui ont notamment collaboré à sa relecture et fait des suggestions toujours pertinentes. À Didier Urli pour des discussions fructueuses portant sur les processus de conception des systèmes d'information, les approches de modélisation des données et les défis que posent l'enseignement des techniques et méthodes sous-jacentes.

Merci à Daniel Pascot de l'Université Laval, expert de renommée internationale en matière de modélisation de données, pour sa banque de cas de modélisation réalisés avec la notation Merise dans le contexte de la méthode Datarun. Certaines études de cas élaborées par M. Pascot et ses collaborateurs nous ont servi d'inspiration pour produire de nouvelles études de cas et des exercices qui mettent en valeur la puissance d'expression de la notation UML pour la modélisation des données et l'analyse des besoins. Ces études de cas portent la mention DP (Daniel Pascot).

Un merci bien particulier à mes étudiantes et à mes étudiants qui ont été appelés à faire les études de cas et à réaliser les exercices. Leurs commentaires m'ont permis de combler quelques lacunes et de corriger les erreurs qui s'y étaient glissées.

Avant-propos

Cet ouvrage est le fruit de mon expérience dans l'enseignement de la conception des bases de données et notamment de la modélisation des données depuis bientôt une vingtaine d'années. Il est l'aboutissement d'une longue réflexion sur l'approche la plus appropriée pour assurer l'initiation à une discipline qui de l'avis de plusieurs est presque aussi ardue que l'apprentissage des mathématiques.

L'analogie avec les mathématiques n'est pas dénuée de pertinence. La modélisation, tout comme les mathématiques, tente de formuler une représentation du monde réel à un très haut niveau d'abstraction, particulièrement sur le plan sémantique. Le non initié est confronté à deux grandes difficultés lorsqu'il s'agit de formuler un modèle de données : la compréhension du problème ou du domaine étudié d'une part et la maîtrise du langage graphique (qui s'exprime à l'aide d'un *formalisme* et d'une *notation*) permettant de formuler une représentation du problème et du domaine.

Cette représentation, d'abord ramenée à l'essentiel puis enrichie de manière incrémentielle, devrait tenir lieu de référence pour une compréhension commune du domaine entre les acteurs impliqués dans un projet de réalisation d'une base de données et des applications qui l'exploitent, qu'ils soient utilisateurs, concepteurs ou administrateurs de celles-ci.

La thèse que sous-tend cet ouvrage est que l'effort didactique est trop souvent mis en aval de la démarche de conception et de réalisation des bases de données (modèle relationnel, algèbre relationnelle, normalisation, langage SQL, etc.) au détriment de la modélisation conceptuelle des données à l'origine de tout projet de conception de base de données. Un modèle conceptuel de données, s'il est réalisé suivant des règles de construction et de validation précises et bien comprises, peut facilement conduire, avec un outil approprié, au schéma physique de la base de données répondant totalement aux exigences du modèle.

C'est la raison pour laquelle nous consacrons par ailleurs un chapitre entier au choix et à l'utilisation des outils qui permettent cette transition directe par la génération de scripts. De plus, de manière à valider conceptuellement la démarche proposée, nous l'illustrons par des exemples concrets d'analyse, de conception et de réalisation d'une base de données mettant en lumière la réalisation incrémentielle de l'application.

À QUI S'ADRESSE CET OUVRAGE

L'ouvrage s'adresse à toute personne qui souhaite être initiée à la modélisation des données et à la conception des bases de données à travers un processus rigoureux valorisant les activités en amont.

- Les étudiants, dont c'est le premier cours sur la conception des bases de données, y trouveront des règles et des astuces permettant de produire de bons modèles conceptuels de données, ainsi que de nombreux exemples illustrant ces règles.
- Les familiers de la modélisation conceptuelle des données qui souhaitent découvrir comment la notation UML permet de supporter le formalisme entité-association avec une capacité de représentation comparable à Merise/2 et supérieure aux diagrammes de Chen.
- Les adeptes de la pro-ingénierie (*forward engineering)* y trouveront une démonstration patente de l'application de ce principe à la conception des bases de données à l'aide de certains outils particulièrement efficaces.

AUTRES OUVRAGES RELATIFS AU SUJET

S'il est vrai qu'il existe de nombreux ouvrages en français qui traitent des systèmes de gestion de bases de données, particulièrement sur les bases de données relationnelles, bien peu d'entre eux consacrent une part importante

aux aspects méthodologiques, notamment à l'analyse des besoins et à la conception des bases de données, ainsi qu'aux divers niveaux de modèles de données proposés notamment par l'American National Standards Institute (ANSI).

L'adaptation en français des éditions récentes des ouvrages américains *Conception et architecture des bases de données*, de Ramez Elmasri et Shamkant Navathe [RamN 04] et *Systèmes de bases de données*, de Thomas Connolly et Carolyn Begg [ConB 05] a donné aux lecteurs francophones deux excellents ouvrages à caractère didactique sur la conception des bases de données.

Le premier ne consacre qu'un seul chapitre aux aspects méthodologiques où il propose une démarche de conception basée sur les trois niveaux d'abstraction des modèles de données. Il introduit la notation UML mais le formalisme entité-association n'y est pas traité. On y évoque brièvement l'existence d'outils de modélisation, dont le Data Modeler de Rational, dans le seul contexte de la réalisation d'un modèle logique de données.

Le deuxième ouvrage consacre plus d'espace aux techniques d'analyse et de conception, au formalisme entité-association, à la notation UML et aux cas d'utilisation pour la définition des besoins en matière de système d'information. Bien que les auteurs évoquent l'existence de pièges lors de l'élaboration d'un modèle conceptuel, qui puissent conduire à des modèles incorrects ou incomplets, ils ne proposent pas de règles précises ou d'astuces permettant aux débutants d'éviter de tels écueils. On n'y traite pas des outils de modélisation disponibles sur le marché.

Côté québécois un troisième ouvrage s'est imposé lui aussi par son caractère didactique : *Système de gestion de bases de données par l'exemple* de Robert Godin [GOD 03]. Il offre certaines similitudes avec l'ouvrage de Connolly et Begg. Comme ce dernier il introduit la notation UML pour la construction de modèle conceptuel et de modèle logique de données ainsi que l'élaboration des cas d'utilisation. L'ouvrage comporte de nombreux exemples mais n'introduit aucune règle pour faciliter au débutant la maîtrise du formalisme entité-association. Un seul outil de modélisation est évoqué, soit ERD de Oracle Designer, pour la construction de modèles conceptuels.

Côté français, *De UML à SQL : Conception de bases de données* écrit par Christian Soutou [SOU 02] fut notre source principale d'inspiration pour la rédaction de cet ouvrage. Notamment pour établir sa facture. Chaque chapitre traite d'un niveau particulier de modèle de données (conceptuel, logique et physique) fidèle en cela avec la méthode Merise/2. Les règles de passage d'un modèle à l'autre y sont traitées abondamment et un chapitre entier est consacré à la comparaison des outils du marché qui automatisent le passage. L'auteur a cependant voulu mettre en parallèle la notation UML avec la

notation utilisée dans Merise/2 pour la représentation du modèle conceptuel. Cela est particulièrement utile pour le modélisateur qui est déjà familier avec l'une des notations et qui souhaite comprendre l'autre. Mais ceci pose une difficulté supplémentaire au débutant qui ne possède ni la maîtrise du formalisme entité-association, ni de l'une des notations utilisées. Soutou ne propose pas de processus d'analyse et de conception, ni de règles formelles pour l'élaboration de modèle conceptuel. Par ailleurs chaque chapitre est truffé d'astuces et de conseils personnels qui sont élégamment mis en évidence grâce aux conventions de mise en page de l'ouvrage dont nous apprécions la grande pertinence et le côté pratique.

GUIDE DE LECTURE

Cet ouvrage s'organise en cinq chapitres. L'introduction développe cet avant-propos. Les chapitres 1 et 2 traitent de modélisation des données. Le chapitre 3 est consacré à la réalisation de la base de données à partir du modèle logique de données. Le chapitre 4 propose une méthode systématique d'analyse et de conception d'une application de base de données. Le chapitre 5 fait une étude comparative des outils logiciels du marché.

Modélisation et conception

L'approche du livre

Notre approche de la modélisation est essentiellement didactique. On expose au chapitre 1 des règles précises de modélisation conceptuelle et des astuces que le modélisateur novice peut utiliser pour le guider dans ses choix. Les règles sont illustrées par de nombreux exemples et des études de cas adoptant la notation UML exclusivement. Ces règles ont notamment pour objectif de réaliser un modèle conceptuel de données dont le modèle relationnel dérivé est normalisé. L'élaboration de ce modèle de haut niveau est ainsi mise en valeur.

Il en va de même pour la dérivation du modèle relationnel à partir d'un modèle conceptuel telle qu'étudiée au chapitre 2. Des règles de dérivation précises sont proposées et illustrées d'exemples et d'études de cas. La notation UML est aussi utilisée pour la représentation graphique du modèle relationnel, assurant ainsi une transition sans heurts vers la maîtrise des concepts du modèle relationnel sans devoir faire appel aux arcanes d'une autre notation.

La réalisation physique d'une base de données est traitée au chapitre 3 en faisant appel au SGBD MS Access, un logiciel bien adapté au débutant qui possède les caractéristiques essentielles d'un SGBD relationnel. Le chapitre 4 propose une démarche systématique de réalisation d'applications de base de données s'inspirant des meilleures méthodes dites *orientées objets* tout en les simplifiant de manière à ce qu'elles puissent être suivies par le débutant. Des exemples complets illustrent la méthode. Chaque étude de cas débute par une analyse des besoins en matière de données, à partir de laquelle l'étudiant est guidé à travers les phases subséquentes de conception et de réalisation de l'application avec MS Access.

Outils sur le marché

Le chapitre 5 permet de voir comment deux outils du marché Sybase PowerAMC et Win'Design permettent de réaliser un modèle conceptuel de données, d'en dériver automatiquement le modèle relationnel, d'optimiser ce dernier et d'en tirer le script nécessaire pour réaliser le modèle physique tout en mettant en évidence la conformité de l'outil aux règles de modélisation et de dérivation introduites aux chapitres 1 et 2.

CONVENTIONS

Cet ouvrage souligne certains éléments clés, qu'ils s'agissent des objectifs d'un chapitre, d'une définition, d'une règle, d'une astuce, d'un principe ou même d'une mise en garde. Le lecteur trouvera en marge gauche un mot clé où une icône précisant la nature de ce que l'auteur souhaite mettre en relief et dans le corps du texte, l'objet de cette mise en évidence.

Lorsqu'il s'agit d'une définition, le terme défini est rédigé en **gras** et si le contenu de la définition réfère à un autre terme défini ailleurs dans l'ouvrage, ce terme sera aussi imprimé en gras. De plus une définition est toujours suivie du terme anglais équivalent placée entre parenthèses. Tous les termes ainsi définis se retrouvent par ailleurs dans l'index. Voici par exemple le format d'une définition.

Modèle de données ▸ Un modèle est une représentation simplifiée d'une réalité. Un modèle de données est une représentation abstraite des données d'un système d'information. Cette représentation est généralement exprimée à l'aide d'un langage graphique appelé **Formalisme** (*Data model*).

Introduction

OBJECTIFS

- Les principes fondamentaux liés à la conception et à l'utilisation des bases de données.
- Comment les systèmes de gestion de bases de données ont évolué pour permettre une gestion efficace et cohérente des données.
- Comment les organisations conçoivent les bases de données et en assurent l'exploitation.
- Pourquoi est-il si important pour les organisations d'élaborer des modèles de données à divers niveaux d'abstraction ?

Qu'on le veuille ou non, les bases de données, tout comme les technologies de l'information de manière générale, sont omniprésentes dans les diverses activités de l'Homme moderne. Nous débutons ce chapitre en évoquant un certain nombre d'applications des bases de données non seulement pour illustrer leur importance centrale dans le fonctionnement de la société et des organisations mais aussi pour montrer les soins particuliers que leurs concepteurs doivent appliquer lors de leur planification et de leur réalisation, afin de servir efficacement les utilisateurs.

Il est hors de question de faire ici un inventaire exhaustif des domaines d'application des bases de données. L'exercice serait long et fastidieux. Il nous importe cependant de mentionner des exemples particulièrement significatifs d'applications conçues pour servir tant les individus que les organisations. Avant de les aborder, nous introduisons trois concepts pour lesquels nous proposons les définitions qui suivent. Ces concepts sont sous-jacents à l'illustration que nous comptons faire dans cette section.

Base de données (BD)

Ensemble structuré d'éléments d'**information**, souvent agencés sous forme de **tables**, dans lesquels les **données** sont organisées selon certains critères en vue de permettre leur exploitation pour répondre aux besoins d'information d'une organisation (*Database*).

Une institution universitaire pourrait par exemple exploiter une seule base de données permettant de gérer l'admission des candidats, d'assurer l'offre de cours à chaque session, d'inscrire les étudiants, de percevoir les frais d'inscription, de compiler les résultats et d'émettre les bulletins de notes.

Application de bases de données

Utilisation de moyens informatiques pour répondre à un besoin déterminé en faisant appel de manière importante à une ou plusieurs bases de données à travers un **système de gestion de bases de données (SGBD)** (*Database application*).

Pour poursuivre avec l'exemple de l'institution universitaire, une application de bases de données pourrait être élaborée uniquement pour la gestion du volet comptable des frais de scolarité. Cette application serait conçue notamment pour émettre les factures pour les frais de scolarité et autres frais afférents, pour percevoir les paiements, rembourser l'étudiant à la suite d'un abandon, suspendre une inscription pour défaut de paiement et le reste.

Système de gestion de bases de données (SGBD)

Logiciel, le plus souvent produit par un éditeur commercial, qui gère et contrôle l'accès à une base de données, assurant ainsi une interface normalisée entre les applications et les bases de données (*Database management system*).

Plusieurs SGBD sont des logiciels commerciaux offerts à grand prix, sous forme de licences d'utilisation sur un serveur, par des éditeurs de logiciel tels que IBM, Oracle, Microsoft, Sybase pour ne nommer que les plus importants qui proposent des SGBD conçus pour exploiter les bases de données de grande envergure. Ces SGBD sont le ***DB2*** de la société IBM, ***Oracle10i*** de la société Oracle ou le ***Microsoft SQL Server***. Certains éditeurs offrent par ailleurs des SGBD bas de gamme, peu coûteux, destinés aux applications de base de données de petites envergures. ***Microsoft Access*** est peut être le mieux connu et le plus utilisé des SGBD de cette catégorie, souvent appelés SGBD *bureautique*. Il existe aussi des SGBD dans le monde du logiciel libre.

Le plus populaire est sans aucun doute ***MySQL*** qui est opéré sous le système d'exploitation ***Linux. MySQL*** n'a rien à envier aux SGBD commerciaux au plan des capacités et des performances.

APPLICATIONS DES BASES DE DONNÉES

Le commerce électronique

L'utilisation des bases de données a connu un essor considérable dans le contexte du développement des échanges commerciaux sous forme électronique. Les entreprises de commerce de détail offrent de plus en plus à leurs clients la possibilité de consulter leur catalogue de produits par le biais de l'Internet de manière à diffuser le prix et la disponibilité de leurs produits et permettre à ces derniers, le cas échéant, de procéder à un achat en ligne. L'accès au catalogue, la possibilité de compléter une transaction d'achat avec autorisation de paiement par carte de crédit ne sauraient être mis en œuvre sans l'utilisation de plusieurs bases de données gérées soit par l'entreprise, soit par une institution financière partenaire. Lorsque le niveau de stock pour un produit atteint un seuil de rupture, le système informatique du commerçant peut émettre sur-le-champ une commande auprès d'un fournisseur par voie électronique. Là encore, une base de données du côté fournisseur va permettre de recevoir et de donner suite à la commande en confirmant une date de livraison, puis en procédant à la facturation le moment venu.

Certaines entreprises comme eBay offrent aux consommateurs la possibilité de faire des échanges commerciaux entre eux selon la formule d'une enchère électronique. La mise en vente ou la gestion des offres en temps réel ne saurait être possible sans une application de base de données sophistiquée qui assure l'impartialité du processus.

Les affaires électroniques

Les affaires électroniques précèdent et prolongent les échanges purement transactionnels liés au commerce électronique pour l'achat, la vente et le paiement des biens et services. Leur domaine d'application est plus large que le commerce électronique. Il concerne aussi bien l'organisation du travail dans une organisation que sa façon de communiquer et d'échanger des données avec ses clients, ses sous-traitants, ses fournisseurs et ses partenaires.

Les institutions financières ont été particulièrement innovatrices sur ce plan mais leurs initiatives n'auront été possibles qu'en adoptant des solutions exploitant de larges bases de données. L'introduction des guichets automatiques par les banques coïncide avec l'adoption de nouvelles technologies de communication, d'une part, et de systèmes de gestion de larges bases de données distribuées, d'autre part, qui permettent au client d'effectuer des transactions sur ses divers comptes bancaires sans égard au propriétaire et à la localisation du guichet utilisé.

Les mêmes bases de données, accessibles traditionnellement via un guichet automatique, le sont aussi par l'Internet ou par des services téléphoniques automatisés.

La plupart des sociétés d'assurance offrent à la clientèle la possibilité d'obtenir en ligne une proposition pour une police d'assurance de dommages sur leurs biens. Elles mettent alors en œuvre des applications de bases de données qui dans un premier temps recueillent auprès du client des données sur la nature des biens et des couvertures souhaitées pour ces derniers. En accédant à des données provenant de sources diverses, tels que des bases de données sur la tarification ou l'historique des réclamations du client, l'application produit sur-le-champ une soumission et, si le client accepte la proposition, une police d'assurance sera émise.

Les institutions publiques et parapubliques ne sont pas en reste sur ce plan. Le citoyen peut soumettre son rapport d'impôt, payer ses taxes scolaires ou municipales par voie électronique, informer le gouvernement d'un changement d'adresse qui sera connu de tous les organismes gouvernementaux concernés. Des initiatives dites de gouvernement en ligne devraient permettre à terme au citoyen de traiter avec le gouvernement grâce à un seul guichet électronique. Ces initiatives ne sont possibles que par la mise en œuvre d'échanges de données entre les centaines de milliers de bases de données administrées par les multiples agences, organismes, sociétés du secteur public.

La gestion électronique des documents

Ce domaine d'application des bases de données concerne la gestion, par des moyens informatiques, du cycle de vie complet d'un document électronique, qu'il soit de nature textuelle, graphique, sonore, vidéo ou logicielle. Ce cycle va de sa création à sa destruction, en passant par sa modification, sa publication, sa diffusion. Cela en vue d'optimiser l'accès à ce document, à l'information qu'il contient ainsi qu'à d'autres documents apparentés.

Dans un tel contexte les bases de données assurent le stockage et la diffusion de données multimédia, souvent qualifiées de *données non structurées*. La gestion électronique des documents a mené au développement de *SGBD objet*. Ce type de système de gestion de bases de données se distingue des *SGBD relationnels* utilisés traditionnellement par les organisations pour le stockage de données structurées sous forme de tables. Il sera question un peu plus loin des caractéristiques de ces divers types de SGBD.

Les bases de données dites *objet* sont centrales au fonctionnement des organisations qui oeuvrent dans le secteur des communications, notamment les entreprises du monde de l'édition, de la production audio-visuelle et de la diffusion. La plupart des chaînes de télévision offrent à leurs auditeurs la consultation, à partir de leur portail Internet, d'un large éventail de documents électroniques. On peut y retrouver notamment la copie textuelle de la retranscription d'une émission, des extraits audio ou vidéo de la même émission ou encore l'émission dans sa version originale intégrale. Tout cela est rendu possible efficacement par la mise en œuvre de bases de données multimédia gérés par un *SGBD objet* ou de type hybride communément appelé *SGBD relationnel-objet*.

Le support à la décision

La décennie quatre-vingt-dix a vu apparaître une catégorie de systèmes d'information permettant la recherche active et l'exploitation, sur le plan décisionnel, de l'ensemble des renseignements stratégiques essentiels qu'une entreprise doit posséder si elle veut faire face à la concurrence et occuper la première place dans son secteur d'activité. Ce type d'application appelé *système de veille stratégique* (connu en anglais sous le vocable de *business intelligence system*) fait appel à une masse considérable de données, provenant de sources multiples recueillies sur une large échelle de temps, regroupées dans une base de données que l'on appelle *entrepôt de données*.

Entrepôt de données ▸ Base de données spécialisée dans laquelle est centralisé un volume important de données consolidées à partir des différentes sources de renseignement d'une entreprise (notamment les bases de données internes) et qui est conçue de manière à ce que les personnes intéressées aient accès rapidement à l'information stratégique dont elles ont besoin (*Data warehouse*).

Un important brasseur américain recueille quotidiennement, à partir de milliers de points de vente, des données sur la vente de ses produits et ceux de ses concurrents. Ces données sont consolidées dans un entrepôt de données qui contient de plus une foule de données démographiques et

socioéconomiques provenant du bureau du recensement. Les analyses menées quotidiennement sur ces données à l'aide de modèles mathématiques sophistiqués permettent au brasseur de cibler la clientèle pour un nouveau produit dans une région donnée, d'établir le moment opportun pour lancer une campagne de promotion sur certains produits, de proposer aux détaillants une présentation de leurs produits favorisant les achats croisés. Cette forme d'exploitation des bases de données est en pleine expansion non seulement dans le secteur du commerce au détail mais aussi dans les secteurs industriels et financiers.

NOTIONS FONDAMENTALES EN MATIÈRE DE GESTION DE DONNÉES

Nous allons aborder dans cette section certains concepts fondamentaux du domaine de la gestion de données. Il importe notamment de définir les termes du vocabulaire courant qui dans le contexte du traitement électronique des données prennent une signification précise.

Donnée et information

Les termes *donnée* et *information,* bien que considérés comme des quasi synonymes dans le vocabulaire de tous les jours, seront traités de manière fort différente dans cet ouvrage.

Donnée ▸ Représentation d'un élément d'information, tel qu'un chiffre ou un fait, codé dans un format permettant son stockage et son traitement par ordinateur (*Data*).

Lorsqu'un client se présente à un point de vente pour payer un produit, la lecture du code barre donne accès à des données liées à ce produit, dont son prix et son nom. Dans ce cas le code barre est une donnée au sens défini ci-dessus puisqu'il s'agit de la représentation codée d'une suite de chiffres, le *code universel de produit* (CUP). Par ailleurs le prix et le nom du produit auxquels correspond le CUP sont aussi des données qui sont codées dans un format faisant appel à une norme largement adoptée en matière de codage binaire des données pour leur traitement par des ordinateurs. Cette norme porte le nom ASCII (*American Standard Code for Information Interchange*). Le format de codage d'une donnée dépend de son type.

Type de donnée ▸ Nature du codage utilisé pour représenter une donnée élémentaire et les opérations applicables à cette donnée. Les types les plus courants sont : entier, réel, texte, date, image, etc. (*Data type*).

Une donnée est une représentation. C'est pourquoi dans sa définition on réfère à la codification, donc à la *syntaxe*. Une information est davantage une interprétation de données dans un contexte particulier. Sa définition fait plutôt référence à la signification d'une donnée mais plus généralement d'un ensemble de données, c'est-à-dire à la *sémantique*, comme le montre la définition suivante.

Information ▸ Une information est une donnée ou un ensemble de données qui a ou ont été interprétée(s) (*Information*).

Les deux concepts sont intimement liés. Dans l'exemple précédent nous référions aux données d'une transaction de vente. La transaction, en plus du nom du produit, de son prix et de son CUP, pourrait comporter le numéro du magasin où l'achat a été effectué. Si toutes les transactions de vente d'une grande chaîne de magasins sont enregistrées dans une même base de données, il sera possible en faisant un traitement statistique sur ces données de savoir quels produits se vendent le mieux dans les divers magasins de la chaîne. Le résultat de ce traitement et l'interprétation qu'en donneront les dirigeants représentent de l'information. Cette information pourrait mener à des décisions, notamment de retirer de ses tablettes certains produits dans certains magasins pour faire place à des produits qui se vendent mieux. Nous dirions en conclusion que l'information induit assez souvent une action, soit une décision dans l'exemple qui nous occupe. Une masse de données, à moins qu'elle ne subisse un traitement approprié et qu'on en retire de l'information, induit rarement une action. Par analogie nous pourrions conclure que les données constituent la matière brute à partir de laquelle l'information est produite.

La vocation première d'une base de données est le stockage des données. Il s'agit donc d'assurer par le biais de cette technologie la *persistance* des données dans le temps. Les systèmes informatisés qui sont mis en œuvre pour le traitement des données peuvent être de simples outils de cueillette de données à la source. On les appelle *systèmes de traitement de transactions* (STT). De plus, ils peuvent être des systèmes qui traitent les données recueillies par les STT pour produire des informations utiles à la prise de décision. Cette deuxième catégorie de système fait appel tant à des ressources informatiques (équipement, logiciel, données) qu'à d'autres types de ressources telles que du personnel ou des procédures administratives. De tels systèmes portent le

nom de *système d'information*, ou encore de *système d'information organisationnel* lorsque le système est conçu pour répondre aux besoins en matière d'information pour l'organisation dans son ensemble.

Système d'information organisationnel

Système constitué des ressources humaines, des ressources informatiques (équipement, logiciel, données) et des procédures permettant d'acquérir, de stocker, de traiter et de diffuser les éléments d'information pertinents au fonctionnement d'une entreprise ou d'une organisation (*Management information system*).

Les systèmes d'information modernes exploitent tous une ou plusieurs larges bases de données à l'aide d'un ou de plusieurs systèmes de gestion de base de données. Les données stockées dans ces bases de données assurent le lien entre d'une part les ressources non informatiques que sont les utilisateurs et les procédures qu'ils doivent appliquer à l'intérieur du système et d'autre part les ressources informatiques. Les données agissent à la manière d'un pont qui relie les ressources informatiques et non informatiques d'un système d'information.

Caractéristiques des systèmes de gestion de bases de données (SGBD)

On s'entend généralement pour dire qu'un logiciel de gestion de données, pour qu'il puisse porter le nom de *système de gestion de bases de données*, doit posséder un certain nombre de caractéristiques fondamentales. Nous les regroupons ici en six catégories.

Indépendance entre les données et les applications

Comme nous le verrons à la section suivante, qui traite des origines et de l'évolution des SGBD, les premières applications informatiques exploitant de grande quantité de données étaient basées sur des fichiers. Les programmes informatiques devaient dans ce contexte comporter une description détaillée des données stockées dans ces fichiers, par exemple le type d'une donnée, sa taille ou son format. Les applications étaient alors fortement dépendantes de la structure des données des divers fichiers qu'elles utilisaient. Un SGBD doit permettre à un programmeur de développer une application sans avoir à encoder dans les programmes les aspects structurels des fichiers d'une base de données. De la sorte, si la structure de la base de données devait être changée, les programmes n'auront pas à être modifiés et, si cela était nécessaire, les modifications ne seraient que mineures. Cette caractéristique des SGBD permet une très grande productivité du personnel qui réalise des applications de base de données.

Contrôle centralisé des données pour éviter toute redondance

Le SGBD doit intégrer dans un même espace de stockage plusieurs fichiers, de manière à éviter toute redondance, c'est-à-dire courir le risque qu'une même donnée se retrouve dans deux ou plusieurs fichiers. Il n'est pas souhaitable qu'une donnée, par exemple l'adresse d'un client, se retrouve dupliquée dans plusieurs fichiers. Une telle situation pose des problèmes au moment où la donnée doit être mise à jour. Si l'adresse d'un client devait changer, il y a un risque que le changement ne soit enregistré que dans un seul fichier. Comment savoir alors quelle est l'adresse la plus récente, celle conservée dans le fichier A ou dans le fichier B ? Même si le changement d'adresse était enregistré dans tous les fichiers où elle est stockée, cela pose un problème au plan de la performance : pourquoi faire plusieurs fois ce qui logiquement ne devrait être fait qu'une seule fois ?

Partage des données et accès concurrents

Le besoin de centraliser le stockage des données de manière à assurer l'unicité de la saisie et de la validation des données a pour corollaire la mise en place de mécanismes techniques qui vont assurer aux utilisateurs un partage équitable de ces données, sans risque de concurrence, mais qui vont aussi empêcher les accès non autorisés. Le SGBD doit posséder un mécanisme d'accès concurrents à une base de données par plusieurs utilisateurs autorisés. Ce mécanisme doit éviter que des opérations simultanées mènent à des incohérences. Considérons par exemple deux utilisateurs qui tentent de modifier le solde d'un compte client. Le SGBD doit identifier, sur la base du principe « premier arrivé premier servi », quel utilisateur aura la priorité pour faire la lecture du solde. Il permettra au second utilisateur la lecture du solde quand le premier utilisateur aura complété sa modification. Ce mécanisme porte le nom de *verrou*. Il interdit tout accès à une donnée ou à un ensemble de données, tant qu'un autre utilisateur procède à leur traitement.

Gestion de la cohérence et de l'intégrité des données

Les SGBD doivent pouvoir gérer le processus de mise à jour d'un ensemble de données comme un tout indissociable pour garantir la *cohérence interne* de la base de données. Dans le contexte de la gestion des comptes clients, supposons que l'application doive conserver dans la base de données à la fois les données de la transaction affectant le solde du client ainsi que la donnée sur le nouveau solde lui-même. Il est hors de question que, par suite d'une panne de l'ordinateur, les données de la transaction aient été

enregistrées mais que le solde n'ait pu être recalculé. Ou à l'inverse que le nouveau solde ait été stocké mais que les données de la transaction n'aient pu être conservées. Un SGBD doit offrir un mécanisme qui fait en sorte que, si une étape d'une opération ne peut être complétée, toute l'opération doit être annulée automatiquement. Il est du ressort du programmeur de l'application de déterminer quelles sont les étapes indissociables d'une même opération et d'appliquer le mécanisme approprié. Ce mécanisme assure *l'atomicité d'une transaction*. C'est-à-dire qu'à la manière d'un atome, toutes les composantes d'une opération, ou d'une transaction, doivent être considérées comme un tout indissociable. Tout comme l'application d'un verrou sur une donnée, le mécanisme est totalement transparent à l'utilisateur car il est mis en œuvre par le programmeur.

L'intégrité des données réfère à la nécessité de voir appliquer automatiquement par le SGBD des contraintes sur la validité d'une donnée. Là encore, il s'agit d'une responsabilité qui incombe aux concepteurs d'une base de données.

Contrainte d'intégrité des données

Ensemble de règles, définies par le concepteur d'une base de données, qui devront en tout temps être respectées par les données de la base de données. Ces règles sont gérées par le SGBD qui en assure l'application et informe l'utilisateur lorsque l'une d'elles est transgressée (*Integrity contraint*).

Des *contraintes d'intégrité des données* sont définies au moment de la conception d'une base de données afin d'écarter, dès leur saisie, les valeurs jugées inacceptables. Ces contraintes permettent donc de mettre en œuvre une validation automatique des données. Le concepteur d'une base de données peut non seulement établir pour une donnée son type ou sa taille, mais fixer par ailleurs une plage de valeurs acceptables, une limite inférieure, ou enfin une limite supérieure. Ces exemples de contraintes sont du type *contrainte d'intégrité sémantique* car elles concernent la signification d'une donnée, donc l'interprétation unique et non ambiguë qui doit être accordée à cette donnée. Il existe un autre type de contrainte dite *contrainte d'intégrité référentielle* qui concerne les valeurs que peuvent prendre deux données associées dans la même base de données. Les SGBD relationnels, comme nous le verrons plus loin, peuvent assurer la satisfaction de contraintes d'intégrité référentielle en faisant en sorte que la valeur prise par une donnée placée dans un champ d'une première table doit être obligatoirement une des valeurs prises par un des champs d'une deuxième table.

Description des données stockées sous forme de métadonnées

La structure d'une base de données, telle que définie spécifiquement par le concepteur pour être prise en charge par un SGBD, est appelée le *schéma physique* de la base de données.

Schéma physique d'une BD ▸ Collection des composants constitutifs de la structure d'une BD, notamment les propriétés des données, les domaines des données (types de données), les fichiers, les contraintes d'intégrité, associés d'une manière ou d'une autre les uns aux autres. Ce schéma est défini par le biais d'un **langage de définition de données** faisant partie intégrante du SGBD (*Database physical schema*).

Le schéma physique est conservé dans la base de données sous forme de *métadonnées* au même titre que les données elles-mêmes. C'est précisément la nature réflexive ou autodescriptive des SGBD qui assure l'indépendance entre les données et les applications dont nous traitions plus haut. Un SGBD doit pouvoir en effet gérer non seulement des données mais aussi des métadonnées.

Métadonnées ▸ Données à propos des données qui sont stockées dans une BD (*Metadata*).

Gestion de la sécurité

La sécurité dans les SGBD possède plusieurs facettes. La première concerne les privilèges d'accès aux données accordés aux utilisateurs. Un SGBD doit permettre d'accorder des privilèges aux utilisateurs jusqu'au niveau d'une donnée pour des opérations de lecture, d'insertion, de mise à jour ou de suppression. Toutefois, en pratique, les privilèges sont généralement accordés pour un groupe d'utilisateurs, à un ensemble de données, soit en lecture (accès), soit en écriture (ajout et modification) c'est-à-dire pour les deux modes d'accès principaux aux enregistrements. Ces privilèges sont accordés par la personne à laquelle l'organisation a confié le rôle *d'administrateur de bases de données*. Les privilèges sont conservés sous forme de métadonnées dans la BD et sont rattachés au code d'accès et au mot de passe que l'administrateur a attribué à l'utilisateur.

Administrateur de bases de données (ADB) ▸ Personne responsable de la réalisation physique de la BD sous forme de schéma physique, du contrôle de la sécurité et de l'intégrité, de la maintenance du SGBD et de la garantie de performance des applications de bases de données qui doit être assurée aux utilisateurs (*Database administrator*).

Il est aussi de la responsabilité de l'administrateur de bases de données de déterminer quelles seront les données qui devront être chiffrées, c'est dire rendues illisibles à ceux qui se seraient procuré frauduleusement des codes d'accès et pourraient en faire un usage inapproprié. Il va de soi que toutes les métadonnées relatives aux droits d'accès, aux codes d'utilisateur, pour ne mentionner que ceux là, doivent être stockés chiffrés dans la BD.

Un deuxième volet de la sécurité concerne la sauvegarde de copies de sécurité d'une BD et la récupération des données à la suite d'une panne. Certains SGBD maintiennent en tout temps une *copie miroir* d'une BD et ce à la volée. C'est à dire que tout changement à une BD est reflété en temps réel dans une deuxième BD qui maintient une copie de la première souvent sur un site physiquement distant. La copie miroir peut par ailleurs être générée en différé par un mécanisme appelé *synchronisation*. Il s'agit d'alimenter, en temps opportun, une BD miroir incluant tous les changements effectués depuis la dernière synchronisation, par exemple au cours de la nuit quand les opérations sont suspendues.

Enfin, un SGBD doit pouvoir conserver une trace de toutes les opérations effectuées sur une BD, sous forme d'un *journal de transactions*. Ce journal complété automatiquement, donc sans intervention humaine, va permettre de reprendre le cas échant certaines transactions qui n'ont pu être complétées à cause d'une panne, et ce, en toute transparence pour les utilisateurs. Le même journal est aussi un outil de vérification et de contrôle pour l'administrateur de la base de données.

Origine et évolution des SGBD

Les systèmes basés sur des fichiers

On ne saurait parler de l'origine des SGBD sans évoquer les premiers systèmes de traitement électronique des données basés sur des fichiers. Cela pour trois raisons principales :

- Ces systèmes ont permis de définir certains concepts toujours en usage même dans le contexte des SGBD les plus évolués, les notions de *fichier de données*, d'*enregistrement*, de *champ* et de *type de données*.
- Bien que cette approche soit maintenant obsolète, il existe encore de nombreux systèmes de ce genre dans les organisations qu'il serait trop coûteux de remplacer par des applications de base de données. On les appelle *systèmes hérités* (ou en anglais *legacy systems*) car ils nous sont légués par des technologies de première génération, dépassées bien sûr,

mais qu'il faut continuer d'utiliser tant que de nouveaux systèmes en cours de développement, ou prévus à l'intérieur d'un plan directeur de développement, ne les auront pas définitivement remplacées.

- Les systèmes basés sur des fichiers comportaient de nombreuses lacunes qui ont permis d'établir de manière empirique les caractéristiques fondamentales des SGBD modernes dont nous avons fait état à la section précédente.

Système basé sur des fichiers

▸ Ensemble de programmes d'application qui exploitent ses propres fichiers de données pour répondre aux besoins spécifiques d'un groupe d'utilisateurs finaux (*File-based system*).

Les premiers grands systèmes de traitement électronique des données élaborés dans les années 1950 et 1960, tels que les systèmes de traitement de la paie, les systèmes de gestion financière et comptable ou les systèmes de gestion des ressources humaines, étaient des systèmes basés sur des fichiers. Ces systèmes étaient conçus pour les besoins spécifiques d'une unité administrative et conséquemment les programmes d'application et les fichiers étaient généralement à l'usage exclusif du personnel de l'unité. Applications et fichiers formaient un ensemble fortement intégré dont l'exploitation était réservée à un groupe exclusif d'utilisateurs. Les concepteurs de ces applications faisaient dès lors appel à des technologies basées sur un certain nombre de concepts dont nous ferons ici état.

Les *fichiers* sont constitués *d'enregistrements* eux-mêmes décomposables en *champs*, chaque champ permettant le stockage d'une donnée. Ces concepts fondamentaux sont explicités à travers les définitions qui suivent.

Fichier de données

▸ Ensemble d'enregistrements identifié par un nom, qui constitue une unité logique de stockage de données pour un ordinateur (*Data file*).

Enregistrement

▸ Groupe de données apparentées, structuré et considéré comme un tout. Chaque donnée du groupe occupe un champ de l'enregistrement. Un champ est défini par son nom, une position dans l'enregistrement et le type de données qu'il permet de stocker (*Record*).

Un fichier est généralement représenté dans les ouvrages techniques sous forme d'un tableau comportant en en-tête une suite d'intitulés donnant le nom des champs, sous lequel sont inscrits les enregistrements. La figure 0-1 illustre un petit fichier appelé **Véhicule** comportant trois enregistrements formés de cinq champs : **No de série**, **Fabricant**, **Modèle**, **Année**, **Immatriculation**. Chaque enregistrement possède une donnée par champ et, pour un champ en particulier, son type de données et sa taille sont systématiquement

les mêmes quel que soit l'enregistrement. Dans cet exemple, le champ **Année** ne stocke que des valeurs numériques entières de quatre chiffres. Chaque ligne du tableau représente un enregistrement et tous les enregistrements ont la même structure définie par les propriétés des champs qui le composent.

FIGURE 0-1 **Fichier *Véhicule***

No de série	Fabricant	Modèle	Année	Immatriculation
YG100P9065QZ84	Ford	Taurus	2005	WWP 657
JK92876T6753W9	Nissan	Pathfinder XE	2004	KDF 324
PK8750927GH786	BMW	320 SI	2002	BGH 629

Les bases de données hiérarchiques et réseaux

Les problèmes liés à l'exploitation des systèmes basés sur des fichiers, notamment la forte intégration des applications et des fichiers, la redondance des données, la difficulté de partager les données et d'assurer leur intégrité, enfin la piètre sécurité des données, ont donné lieu à un nouveau paradigme en matière de gestion des données, le système de gestion de bases de données.

Une première génération de SGBD est née de travaux de recherche menés dans les années 1960 pour le compte de la NASA dans le cadre du projet Apollo et commercialisés dans les années 1970 par la société IBM sous le nom ***IMS*** (Information Management System). Le SGBD était basé sur le concept que les enregistrements conservés dans divers fichiers pouvaient être liés selon une certaine hiérarchie et constituer un assemblage arborescent. On a appelé *structure hiérarchique* cette forme de liaison entre des enregistrements de fichiers distincts. En vertu de ce modèle d'organisation des données, un enregistrement peut être le *père* de plusieurs enregistrements qui à leur tour peuvent avoir des *fils*. Par exemple un fichier appelé **Étudiant** contenant des données sur les étudiants peut être lié à un autre fichier, **Cours inscrits**, contenant des données sur les cours suivis par les étudiants et le cas échéant le résultat obtenu. À l'aide d'un langage dit *de navigation* entre les fichiers, le programmeur accédant à un enregistrement du fichier **Étudiant** peut repérer automatiquement les enregistrements *fils* du fichier **Cours inscrits**. Un SGBD hiérarchique possède un avantage indéniable sur les systèmes basés sur des fichiers. Il permet en effet d'établir des liens *un à plusieurs* entre les fichiers d'une base de données : un enregistrement *père* peut être lié à plusieurs enregistrements *fils*.

Dans la même période, des chercheurs de la société américaine General Electric eurent l'idée de généraliser l'approche des SGBD hiérarchiques en proposant un modèle d'organisation des données permettant des liens

plusieurs à plusieurs entre les fichiers. La hiérarchie père-fils n'existe pas dans ce modèle, car un enregistrement peut avoir plusieurs *successeurs* de même que plusieurs *prédécesseurs*. Les fichiers sont liés en *réseau maillé* à la manière des réseaux d'égout où à un point du réseau convergent des connecteurs effluents et des connecteurs affluents. Ces SGBD sont dits *de type réseau* (ou simplement *SGBD réseau*)et permettent de représenter des associations complexes entre les fichiers. Cette catégorie de SGBD a suscité beaucoup d'intérêt dans la communauté des chercheurs et des utilisateurs intéressés par les bases de données. Cela a donné lieu, à la fin des années 1960, à la création d'un groupe de travail visant à définir des normes pour le SGBD de type réseau portant sur les caractéristiques requises pour assurer la création des bases de données ainsi que la manipulation de données. Ces normes, connues sous le vocable ***CODASYL***, décrivent en quelque sorte l'environnement requis pour la conception et l'exploitation des bases de données réseaux. Elles reconnaissaient pour la première fois de manière formelle l'importance du *schéma physique* de la base de données, le rôle clé de la personne responsable de sa construction et de son entretien, c'est-à-dire *l'administrateur de base de données (**ABD**)*. La norme met en évidence la nécessité de mettre à la disposition de l'ABD un langage pour la construction de schéma physique appelé le *langage de définition de données* ainsi que d'offrir au programmeur un langage pour exploiter les données, le *langage de manipulation de données*.

Les SGBD hiérarchiques et réseau constituent la *première génération* de SGBD. Malheureusement ils comportaient certaines lacunes notamment au plan des fondements théoriques. De plus ils ne permettaient pas en pratique d'assurer l'indépendance tant souhaitée entre les applications et les données. En effet, comme le lien entre deux enregistrements était implanté à l'aide d'un pointeur, soit une sorte d'adresse permettant de repérer un enregistrement associé, cela donnait lieu à des programmes complexes même pour des requêtes simples. Les SGBD de première génération auront néanmoins permis de définir de manière détaillée les spécifications de l'environnement des bases de données dont nous allons traiter à la section suivante.

Les bases de données relationnelles

Le début des années 1970 a été marqué par une avancée importante dans le domaine des SGBD. E. J. Codd, chercheur au laboratoire de recherche IBM de San Jose en Californie, s'inspirant de l'algèbre relationnelle, proposa un mode d'organisation des données ne nécessitant pas de pointeurs pour lier les enregistrements. Non seulement ce modèle avait ses fondements théoriques solidement établis dans l'algèbre relationnelle, mais il était aussi d'une grande simplicité. La structure d'un fichier est définie comme une *relation* entre des données provenant d'un nombre fini de *domaines*. Les enregistrements

sont des *tuples*, et constituent des occurrences de la relation. Les liens sont assurés entre deux enregistrements sur la base d'un champ de même type, commun aux deux enregistrements. Si les champs communs possèdent la même valeur, les enregistrements sont logiquement liés. Il n'est dès lors plus nécessaire de gérer des pointeurs physiques pour assurer ces liens. De *physiques* qu'étaient les liens dans les SGBD hiérarchiques ou réseau, les liens sont dorénavant *logiques*, basés sur les valeurs des champs, ce qui rend la navigation entre les enregistrements beaucoup plus souple. Nous reviendrons au chapitre 2 sur le modèle relationnel pour y faire état de ses nombreux avantages et traiter des concepts de clé primaire et de clé étrangère qui permettent de réaliser les liens.

Ce mode d'organisation des données a donné naissance à une pléthore de *SGBD relationnels* commerciaux et non commerciaux dont ***DB2*** d'IBM, ***Oracle10***, ***Microsoft Access***, ***MySQL***, pour ne nommer que les mieux connus. Les SGBD relationnels sont de deuxième génération et ils ont tous en commun intrinsèquement un langage appelé SQL (Structured Query Language). SQL agissant à la fois comme langage de définition et langage de manipulation de données.

Malgré leurs nombreuses qualités, les SGBD relationnels ont un certain nombre de lacunes pour l'expression de modèles de données à un haut niveau d'abstraction, ce que nous définirons au chapitre 1 comme le modèle conceptuel de données. Un effort considérable de recherche et de réflexion a été fait au cours des vingt dernières années pour développer des méthodes et des formalismes permettant d'exprimer un modèle de données dépouillé des contraintes du modèle relationnel mais qui puisse se traduire assez directement dans un SGBD relationnel. C'est l'objectif premier de cet ouvrage que de montrer comment à l'aide de méthodes et d'outils appropriés on peut construire des modèles de données qui soient indépendants du type de SGBD dans lequel ils seront réalisés.

Les bases de données orientées objets et les autres

Le développement de langages orientés objets tels que ***Smalltalk*** et ***C++*** a conduit à la mise au point de SGBD devant assurer la *persistance des objets*, soit le stockage permanent sur un support de mémoire auxiliaire des objets créés à l'aide de tels langages. Ce SGBD dit *orienté objet* ou simplement SGBD *objet*, qui a connu un essor somme toute limité appartient à la *troisième génération*. Le SGBD ***Gemstone*** par exemple est une extension du langage Smalltalk. Le développement des SGBD objet a été freiné par des SGBD hybrides incorporant le modèle relationnel et le stockage d'objets. Les objets peuvent être soit de type structuré comme ceux créés par un langage orienté

objet ou de type non structuré telles que des images, de la vidéo, des trames sonores. Le type non structuré est aussi appelé BLOB (Binary Large OBject). On donne le nom SGBD *relationnel-objet* à cette forme hybride car il combine les propriétés du SGBD relationnel et du SGBD objet. L'éditeur américain Informix a été un précurseur en matière de SGBD relationnel-objet pour être suivi et vite dépassé par Oracle et IBM avec leur produit ***Oracle8*** et ***DB2 Universal Database System*** respectivement.

À l'aube du XXIe siècle on parle d'une *quatrième génération* de SGBD. Il s'agit d'une catégorie hétérogène de SGBD conçus avant tout pour des applications spécialisées, comme par exemple et non exclusivement, les SGBD OLAP (*On Line Analytical Processing*) largement utilisés pour l'entreposage des données et l'exploration des données, les SGBD XML pour les applications Web, ou enfin les SGBD de contraintes pour les applications d'optimisation.

ENVIRONNEMENT DE BASES DE DONNÉES

Comme nous l'avons vu à la section relatant les caractéristiques des SGBD, une base de données est par définition partagée par plusieurs utilisateurs finaux. Elle doit en conséquence satisfaire leurs besoins en information quel que soit l'utilisateur. *L'utilisateur final* est un élément central de l'environnement d'une base de données car il en est de fait le client. Mais il n'est pas la seule personne jouant un rôle majeur dans cet environnement. Nous avons évoqué plus tôt le rôle joué par *l'administrateur de bases de données*. Notons brièvement le rôle d'autres individus qui interviennent dans l'environnement d'une base de données.

- *Les concepteurs de bases de données*. Leur rôle consiste à élaborer le design d'une base de donnée. Ils traitent directement avec les utilisateurs dans une organisation pour identifier les données et les contraintes sur les données qui devront être conservées dans une BD. Les concepteurs sont aussi appelés des *modélisateurs de données* car leur travail consiste en outre à élaborer des *modèles de données*, soit des plans de plus en plus détaillés des données et des contraintes sur les données de la BD. Comme nous le verrons plus loin, leur travail consiste à élaborer des modèles du plus général au plus spécifique. Le modèle le plus général est le *modèle conceptuel des données*, indépendant du type de SGBD cible. Il ne fait état que des entités, leurs attributs, des associations et contraintes sur les données. Les contraintes sur les données sont généralement exprimées sous forme de *règles de gestion* des données. Un modèle de données plus spécifique est le *modèle logique de données* qui est élaboré en fonction d'un type de SGBD

tel que relationnel, réseau, hiérarchique ou objet. Enfin le *modèle physique de données* est la transposition d'un modèle logique dans le SGBD adopté pour implanter la BD et l'application de base de données.

- *Les développeurs d'applications*. Il s'agit essentiellement des personnes qui interviennent pour définir l'architecture et réaliser les programmes d'application qui vont fournir aux utilisateurs les fonctionnalités pour accéder à la base de données dont le modèle physique vient d'être réalisé. Ce sont les *architectes*, les *analystes* et les *programmeurs*.

De manière à supporter efficacement le travail des concepteurs de BD et les développeurs d'application de base de données, bon nombre d'éditeurs d'outils de développement ou de SGBD commerciaux adoptent une proposition faite par un organisme de normalisation américain, l'ANSI (*American National Standards Institute*), qui préconise une approche à trois niveaux pour réaliser le design d'une base de données. Bien qu'il ne s'agisse pas d'une norme en bonne et due forme, ce découpage fait désormais référence en la matière et nous l'adoptons d'emblée dans cet ouvrage pour structurer la démarche de conception d'une base de données. Voyons en quoi consiste cette proposition qui a fait l'objet d'un premier rapport en 1975, autour duquel un large consensus s'est développé.

Les niveaux d'abstraction

Le niveau externe

En vertu de la proposition de l'ANSI, les données d'une base de données sont décrites selon trois niveaux d'abstraction. La façon dont un utilisateur perçoit les données s'appelle une *vue*. Chaque utilisateur, selon ses besoins et ses prérogatives, peut avoir une vue différente sur les données stockées dans une BD. Ce niveau d'abstraction est appelé le *niveau externe*. Il s'agit du plus haut niveau d'abstraction.

Le niveau interne

Au plus bas niveau d'abstraction se situe le *niveau interne*. Il représente la manière dont le SGBD perçoit les données stockées dans une BD. La description des données au niveau interne est donnée par deux modèles : le *schéma logique des données* et le *schéma physique des données*. Ces modèles sont fortement dépendants du type de SGBD choisi pour mettre en œuvre la base de données.

Le niveau conceptuel

De manière à assurer une totale indépendance de la description des données, soit en regard d'un groupe d'utilisateurs soit d'un SGBD en particulier, une couche intermédiaire doit exister et on l'appelle le *niveau conceptuel*. À ce niveau le *schéma conceptuel de données* va décrire un domaine de données à prendre en charge par une BD sans égard à un utilisateur en particulier et il sera relativement stable dans le temps. Cela signifie qu'un changement à l'organisation physique d'une BD ou encore le choix d'un autre SGBD n'aura aucun impact sur les vues déjà offertes aux utilisateurs. Par ailleurs l'ajout de nouvelles vues ou d'un nouveau groupe d'utilisateurs avec de nouvelles vues ne devrait avoir aucun impact sur l'organisation physique des données. Le niveau conceptuel assure ainsi une couche *d'isolation* entre les besoins des utilisateurs et l'organisation physique de la BD. Chaque vue est en fait un sous-ensemble du schéma conceptuel et est quelques fois appelée un *sous-schéma*.

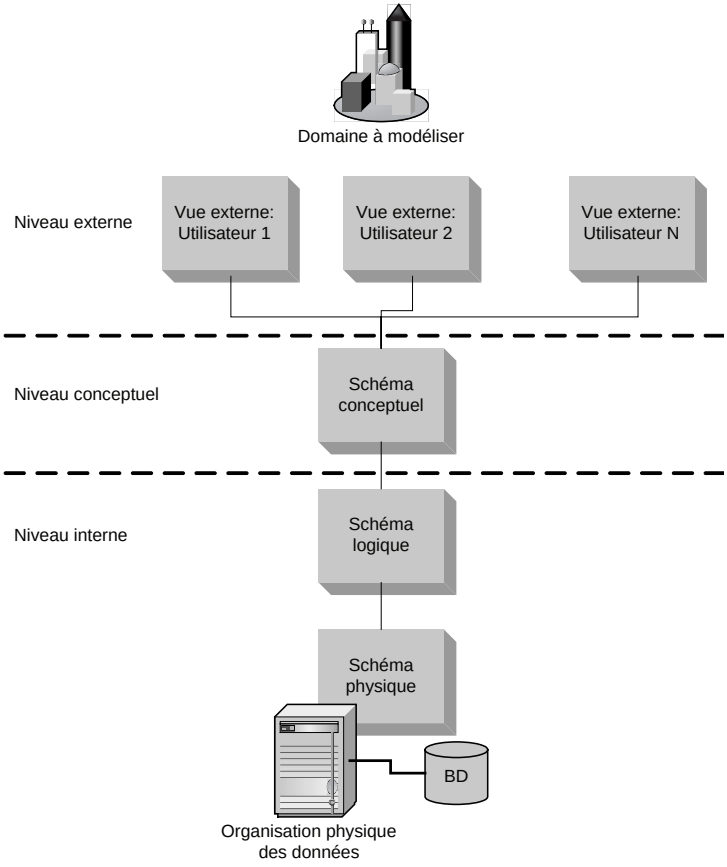
La figure 0-2 présente les trois niveaux d'abstraction sur les données tels que proposés par l'ANSI. Nous aurons le loisir de revenir aux cours des trois prochains chapitres sur la nature précise de ces divers niveaux d'abstraction ainsi que sur les modèles qui les supportent. Notons en passant que pour rester cohérent avec de nombreuses méthodes de conception de bases de données, nous utiliserons dorénavant les termes *modèle conceptuel de données*, *modèle logique de données* et *modèle physique de données* pour parler des schémas proposés par l'ANSI. En somme dans le cadre de notre ouvrage le terme *schéma*, tel que défini dans le rapport de l'ANSI, est synonyme de *modèle*.

Les trois niveaux d'abstraction et les modèles qui en découlent offrent un cadre rigoureux de description des données pour les concepteurs d'une base de données. L'élaboration du modèle physique de données et la réalisation des applications de base de données exigent par ailleurs des outils et des langages dont les développeurs font usage. Les travaux du groupe ***CODASYL*** ont permis d'identifier deux types de langage, destinés surtout aux développeurs mais aussi aux utilisateurs experts, qui devraient être présents dans l'environnement d'une base de données. Il s'agit du *langage de définition de données (LDD)* et du *langage de manipulation de données (LMD)*.

Langages de bases de données

En pratique cependant on ne retrouve dans un SGBD qu'un seul langage qui comporte à la fois des instructions pour la définition de données et d'autres pour la manipulation des données. Ce langage est en quelque sorte constitué de deux ensembles d'instructions ayant leur vocation propre. De plus, bien que les instructions écrites dans ce langage puissent être exécutées directement

FIGURE 0-2 **Les trois niveaux d'abstraction de l'ANSI**



par le SGBD par l'intermédiaire d'un module appelé *interprète*, elles sont généralement enchâssées par le programmeur dans un programme réalisé avec un langage de haut niveau tel que C, C#, C++, Java ou Visual Basic, puis *compilées* avec les autres instructions du programme pour produire le programme d'application exécutable.

Langage de définition de données (LDD)

Langage qui permet au concepteur du **schéma physique** ou à l'**administrateur de la base de données** de nommer les entités, les attributs et associations constituant la structure de la BD, ainsi que les contraintes d'intégrité et de sécurité associées (*Data Definition Language*).

Le schéma physique est décrit à l'aide d'un *script*, c'est-à-dire une séquence d'instructions du LDD ne comportant ni conditions, ni itérations contrairement aux programmes écrits dans un langage de haut niveau. Le LDD est donc par définition *non procédural*. Le script suivant, écrit pour un SGBD relationnel dans le langage SQL, permet de créer l'entité **Ami** (une table dans le modèle relationnel) avec ses attributs et d'ajouter une contrainte sur l'attribut **RéfAmi** qui en fait la clé primaire de la table.

```
CREATE TABLE Ami
([RéfAmi] integer,
[Nom] varchar (30),
[Prénom] varchar (30),
[DateNaissance] date,
[Téléphone] varchar (10),
[Remarques] memo,
CONSTRAINT [Index1] PRIMARY KEY ([RéfAmi]);
```

La compilation des instructions d'un script de LDD produit des *méta-données* qui seront conservées avec la BD dans ce qu'il est convenu d'appeler le *catalogue système*.

Langage de manipulation de données (LMD)

Langage qui fournit un ensemble d'opérations élémentaires de manipulation de données telles que l'insertion, la modification, la recherche, l'extraction et la suppression de données contenues dans une base de données (*Data Manipulation Language*).

Alors que le langage de définition de données est réservé aux spécialistes, concepteurs ou administrateurs, le langage de manipulation de données est très souvent utilisé, directement ou indirectement, par les utilisateurs finaux. Les instructions du LMD qui permettent la recherche et l'obtention de données constituent un *langage de requête* (*query language*). On réserve généralement le terme *requête* pour une instruction écrite dans un langage de requête pour la recherche et l'extraction de données dans une BD. Cette distinction peut paraître étonnante au novice car le langage SQL (*Structured Query Language*), malgré ce que laisse croire son nom, est un langage qui combine à la fois des instructions de définition et de manipulation de données. Sur le plan de la manipulation de données, il se présente non seulement comme un langage de requête au sens strict mais offre de plus des instructions pour la suppression, l'insertion ou la modification des données. SQL est donc un LMD complet, couplé d'un LDD.

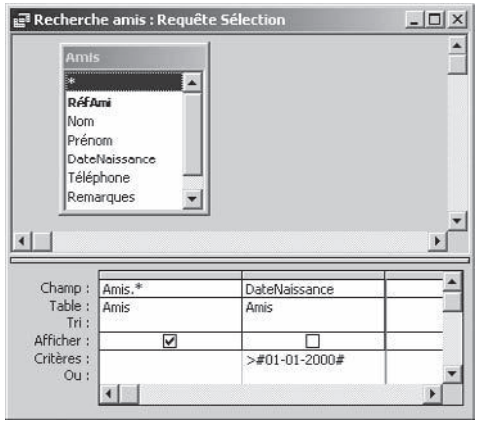
Voici exprimé dans le langage SQL une instruction pour la recherche et l'extraction des enregistrements de la table **Ami** dont la valeur du champ **DateNaissance** est strictement supérieure au premier janvier 2000.

```
SELECT Ami.*
FROM Ami
WHERE (((Ami.DateNaissance)>#1/1/2000#));
```

Plusieurs SGBD relationnels offrent désormais une interface graphique pour faciliter à l'utilisateur la tâche de rédaction d'une requête. Il s'agit en fait de véritables langages de requête qui permettent de construire une requête par l'exemple. Un représentant bien connu de cette catégorie est le QBE (*Query-by-Example*) intégré au SGBD Microsoft Access.

Ainsi, la requête exprimée ci haut en SQL peut être aussi définie à travers une grille comme le montre la figure 0-3. Tout comme SQL, QBE permet aussi de formuler des requêtes pour la suppression, l'insertion ou la modification d'enregistrements. L'utilisateur n'a donc plus à se plier à une syntaxe souvent rébarbative pour formuler sa requête. Sa tâche consiste alors essentiellement à compléter les paramètres de la requête à travers une grille.

FIGURE 0-3 **Requête de sélection exprimée graphiquement avec QBE**



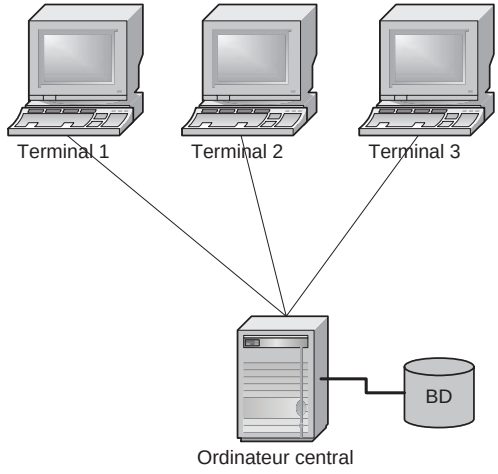
Architecture des SGBD multiutilisateurs

La première architecture mise en œuvre pour permettre à plusieurs utilisateurs d'accéder concurremment à une base de données est celle du *télétraitement*. Ce terme met en évidence le fait que toute la capacité de traitement des données est centralisée sur un ordinateur central auquel les utilisateurs accèdent par le biais de simples terminaux. Ces terminaux sont du type *passif* car ils n'interviennent nullement dans le traitement des données, ne servant qu'à transmettre, recevoir et disposer à l'écran les données reçues.

Une telle architecture impose à l'ordinateur central une charge de traitement considérable car ce dernier doit supporter non seulement les tâches d'accès à la base de données et d'exécution de l'application de base de données mais en outre celle de mettre en forme des données transmises aux terminaux qui n'auront qu'à les afficher selon les directives reçues. Cette architecture impose aux organisations l'exploitation et l'achat coûteux d'ordinateurs de grande taille.

La figure 0-4 illustre cette architecture qualifiée aussi de *traitement centralisé*.

FIGURE 0-4 **Architecture de traitement centralisé**



Le développement des ordinateurs personnels peu coûteux ainsi que des réseaux de communication à haut débit a permis l'élaboration d'une nouvelle architecture misant sur la capacité de traitement dévolue localement aux *postes de travail* des utilisateurs. La capacité de traitement n'est plus désormais l'apanage de l'ordinateur central. Les utilisateurs disposent alors de postes de travail qui prennent en charge en tout ou en partie le traitement des données. Cette architecture est dite à *traitement distribué* par opposition à la forme traditionnelle de traitement centralisé. La forme la plus connue de traitement distribué est *l'architecture client-serveur*.

Architecture client-serveur

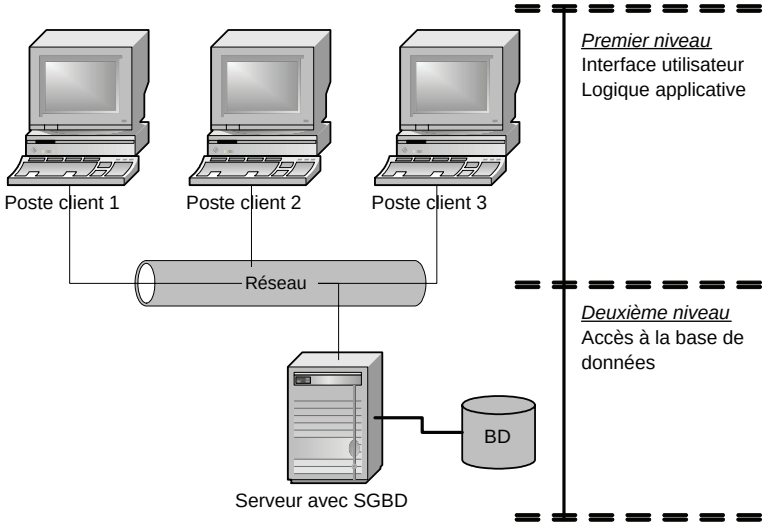
Spécification d'un système informatique dans lequel un processus appelé le ***serveur*** agit comme fournisseur de ressources pour d'autres processus qui demandent ces ressources, soit les processus ***clients***. Le processus client et le processus serveur s'exécutent le plus souvent sur des machines différentes reliées au même réseau (*Client-Server Architecture*).

L'architecture client-serveur vise à décharger l'ordinateur central du plus grand nombre de tâches possible. La forme la plus simple que prend cette architecture est le modèle à deux niveaux (en anglais *two-tier*) illustré à la figure 0-5. Dans les applications de base de données, on retrouve généralement trois composants principaux : la *logique de présentation (interface utilisateur)*, la *logique applicative* aussi appelée la logique métier et la *logique d'accès aux données*.

La logique de présentation est responsable de la gestion de l'interface utilisateur notamment la présentation des données à l'écran et l'interaction avec l'utilisateur. La logique applicative prend en charge les traitements propres à l'application, soit les règles d'affaires d'un métier, la nature des calculs qui doivent être effectués. La logique d'accès aux données est relative à l'exécution des requêtes par le SGBD pour fournir à la logique applicative les données dont elle a besoin pour exécuter les processus du métier.

En vertu de l'architecture client-serveur *à deux niveaux*, le poste de travail de l'utilisateur, appelé le *poste client*, prend en charge tous les traitements liés à la fois à la logique de présentation et à la logique applicative. La logique d'accès est dévolue à un serveur qui agit comme serveur de données exclusivement. Le poste client compose des requêtes, dans le langage SQL par exemple, selon les besoins en données de l'application qui s'exécute sur le poste. Il transmet ensuite la requête à travers le réseau au serveur qui l'exécute à son tour et retourne en réponse au poste client le résultat de son exécution.

FIGURE 0-5 **Architecture client-serveur à deux niveaux**



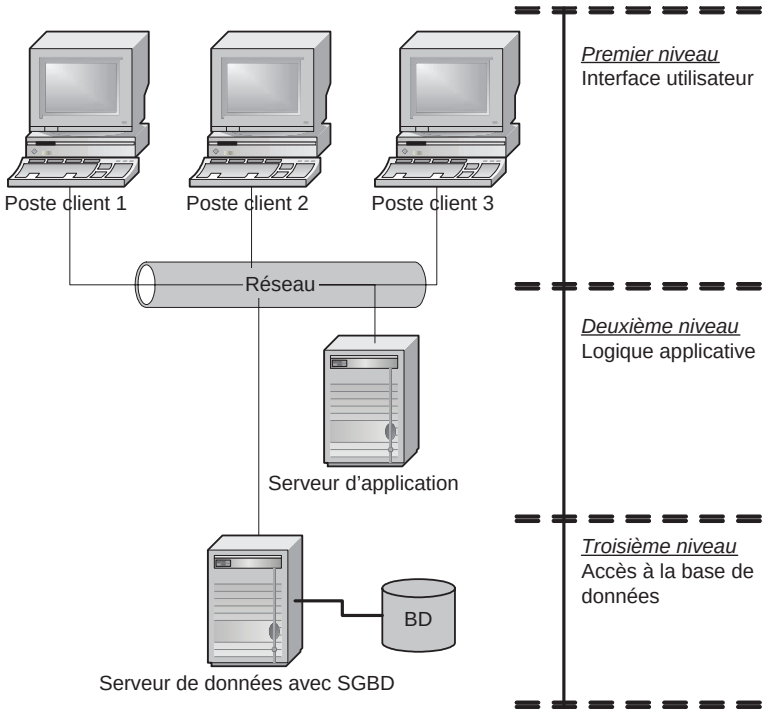
Une telle architecture permet aux organisations de se doter de ressources informatiques centrales de taille modeste tout en misant sur les ordinateurs personnels des utilisateurs, par ailleurs utilisés pour des tâches de bureautique, pour l'exécution d'applications de base de données. Cet avantage est obtenu au prix d'une administration beaucoup plus lourde du parc d'équipement, d'une plus grande complexité des applications et en conséquence de frais d'entretien et de mise à jour plus élevés.

De manière à simplifier l'entretien et la mise à jour de la logique applicative, un autre modèle client-serveur a été mis au point permettant de délester le poste client de ce composant. Il s'agit de l'architecture client-serveur *à trois niveaux* (*three-tier*). Dans ce modèle la logique applicative est prise en charge par un deuxième serveur, ne laissant au poste client que les tâches liées à la gestion de l'interface utilisateur. C'est ainsi qu'est né le concept de *client léger*, léger dans le sens que le poste client ne nécessite plus de ressources aussi importantes en matière d'espace de mémoire principale et de vitesse d'exécution que demandait le modèle à deux niveaux. Dans certaines applications client-serveur à trois niveaux, la gestion de l'interface utilisateur se fait avec un simple navigateur Web. Avec pour conséquence une administration considérablement simplifiée des mises à jour des applications côté client. L'entretien de la logique applicative ne concerne désormais qu'une seule machine que l'on appelle le *serveur d'application*. La figure 0-6 illustre cette deuxième forme d'architecture client-serveur.

La facilité d'administration des mises à jour n'est pas le seul avantage d'un modèle à trois niveaux. Il est en revanche d'une grande souplesse. Il permet par exemple d'assurer aux utilisateurs un service sans interruption en cas de panne. En mettant en œuvre ce modèle avec plusieurs serveurs d'application, offrant tous la même logique applicative, la panne d'un serveur a peu d'impact sur les utilisateurs car les autres serveurs pourront prendre la relève en tout temps même si la performance globale s'en trouvera légèrement dégradée. De plus, le serveur d'application peut communiquer avec d'autres serveurs d'applications pour obtenir des ressources et des services spécialisés. Dans une application de commerce électronique par exemple, le serveur qui prend en charge la transaction du client peut faire appel à un autre serveur, spécialisé dans le traitement du paiement par carte de crédit, de manière à compléter l'achat. Le commerçant peut se procurer ce service auprès d'un fournisseur, sans avoir à développer ou à exécuter sur ses propres équipements ce composant du système.

Nous concluons cette section avec une illustration présentée à la figure 0-7 qui schématise l'environnement des bases de données que nous venons de décrire. Cet ouvrage met en lumière les méthodes et les outils exploités par les modélisateurs et les administrateurs pour la conception des

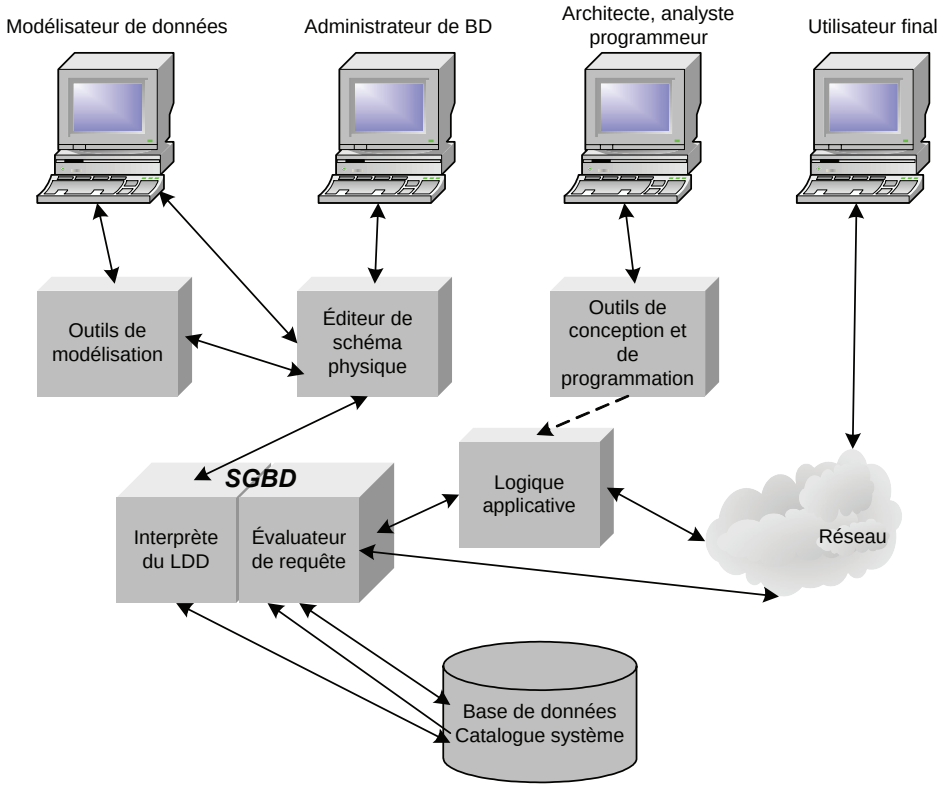
FIGURE 0-6 **Architecture client-serveur à trois niveaux**



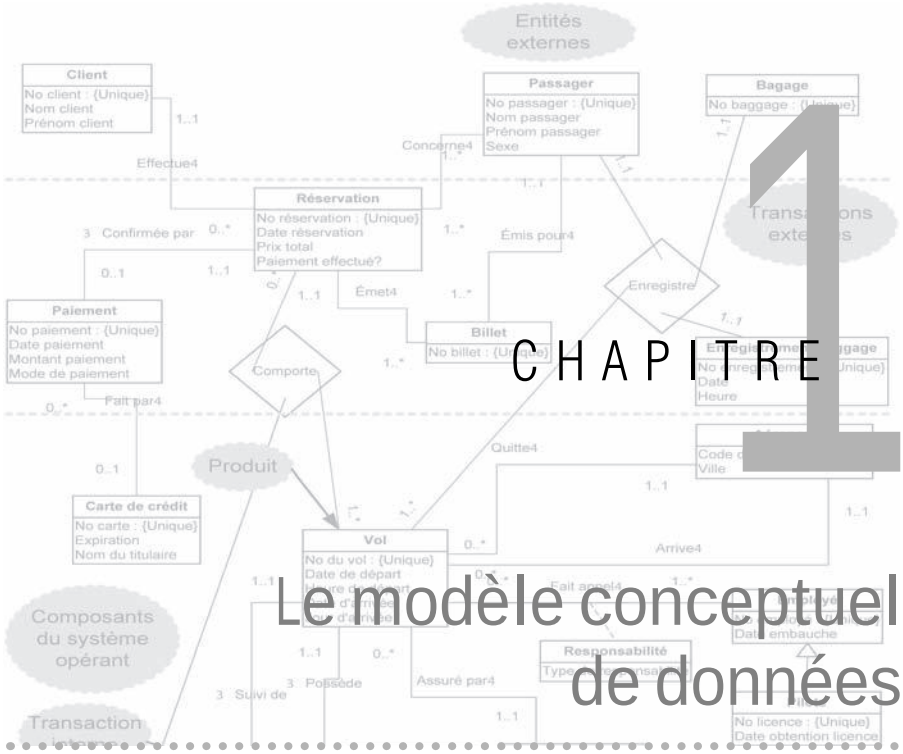
bases de données. Les chapitres 1 à 3 portent sur ce sujet en répondant aux prescriptions de l'ANSI en ce qui a trait aux modèles qu'ils doivent produire et l'ordre dans lequel ceux-ci doivent l'être. Au chapitre 4, nous illustrerons la nature et l'importance de ces modèles dans le contexte de la réalisation d'une application de base de données.

Les trois premiers chapitres traitent respectivement de la modélisation conceptuelle, logique et physique des données. Toutes les méthodes de conception de base de données suggèrent la production de ces modèles dans cet ordre. Le lecteur y trouvera une description détaillée des règles et des formalismes dont font usage les modélisateurs pour réaliser ces modèles, ponctuée de nombreux exemples qui peuvent servir de modèles génériques. Bien qu'il existe des outils qui permettent d'appliquer automatiquement certaines règles, notamment les règles de passage d'un modèle conceptuel au modèle logique, ou encore la génération d'un schéma physique de données à partir d'un modèle logique, il nous apparaît impératif que tout modélisateur de données possède une bonne compréhension de ces règles pour pouvoir,

FIGURE 0-7 **Environnement de bases de données**



le cas échéant, modifier les modèles produits, afin d'optimiser la performance de la BD notamment, sans compromettre la cohérence globale des divers modèles.



CHAPITRE 1

Le modèle conceptuel de données

OBJECTIFS

- Concepts fondamentaux liés au formalisme entité-association (EA).
- Règles et principes assurant l'élaboration d'un modèle conceptuel complet et cohérent.
- Détail de la formulation d'un modèle conceptuel dans la notation UML.
- Notations pour exprimer un modèle conceptuel à l'aide du formalisme EA.

Ce chapitre décrit la première étape dans la réalisation d'une base de données : l'élaboration d'un *modèle conceptuel de données*. Chaque étape de la méthode préconisée dans cet ouvrage produit un modèle de données qui sera affiné à l'étape suivante, jusqu'à la spécification finale de tous les détails d'implantation de la BD pris en charge par un SGBD.

Modèle de données Un modèle est une représentation simplifiée d'une réalité. Un modèle de données est une représentation abstraite des données d'un système d'information. Cette représentation est généralement exprimée à l'aide d'un langage graphique appelé **Formalisme** (*Data model*).

Le modèle conceptuel de données est le seul modèle qui ne découle d'aucun autre modèle. Il est réalisé dans le cadre d'une analyse des besoins portant sur les données requises pour assurer le bon fonctionnement d'une organisation ou d'un secteur spécifique de l'organisation. Cette analyse des besoins est menée auprès des principaux utilisateurs du système d'information et vise à formuler une compréhension commune de la nature des données, de leur sémantique (c'est-à-dire leur *signification*) et de leur utilisation dans une organisation. Le modèle conceptuel de données est en conséquence un modèle non technique compréhensible par tous. Il s'agit d'un outil de communication pour assurer une interprétation commune des besoins exprimés.

Modèle conceptuel de données Un modèle conceptuel de données est une représentation des besoins en matière de données pour un système d'information. Il met en évidence les entités, leurs attributs, les associations et contraintes entre ces entités pour un domaine donné. Cette représentation, de nature sémantique, ne comporte aucune indication concernant la structure de mémorisation des données associées aux entités. Le modèle conceptuel est généralement représenté à l'aide du **Formalisme** entité-association (*Conceptual model*).

Formalisme Un ensemble de règles de représentation permettant de formuler un modèle graphiquement. Il comporte un certain nombre de concepts de base permettant d'exprimer un modèle. La représentation graphique des concepts dépend de la **Notation** employée (*Formalism*).

Plusieurs formalismes ont été proposés pour réaliser un modèle conceptuel de données, par exemple le réseau sémantique de J.-F. Sowa [SOW 84], mais seul le formalisme entité-association fait l'objet d'un réel consensus en cette matière et est une composante essentielle de la plupart des outils de conception de base de données. Malheureusement, ce qui ne fait guère consensus, c'est la *notation* utilisée pour représenter graphiquement les éléments de base du formalisme. Comme nous le verrons plus loin dans cette section, plusieurs notations existent en parallèle à travers divers outils de modélisation. En revanche, l'utilisation de la notation proposée dans le langage de modélisation UML (*Unified Modeling Language*) gagne de nombreux adeptes parmi les éditeurs de logiciels pour la réalisation de système d'information, en particulier pour la formulation d'un modèle conceptuel de données. C'est la raison

première pour laquelle la notation UML a été adoptée dans le cadre de cet ouvrage. Nous reviendrons plus loin sur les autres raisons qui justifient ce choix.

Notation ▸ Convention de représentation graphique des concepts liés à un formalisme, proposée par un ou plusieurs auteurs. Un même **Formalisme**, tel que le formalisme entité-association, peut faire appel à diverses notations. Le choix de la notation est laissé à la discrétion du modélisateur mais il est souvent imposé par l'outil employé par ce dernier pour réaliser le modèle (*Notation*).

Avant de nous attarder aux arcanes de la notation UML, revenons aux concepts de base du formalisme entité-association.

CONCEPTS DE BASE DU FORMALISME ENTITÉ-ASSOCIATION

Le *formalisme entité-association* (EA), initialement appelé entité-relation (*entity-relationship*) par son auteur P. Chen [CHE 76], s'est vite imposé à travers diverses méthodes de conception de bases de données comme le formalisme de choix pour exprimer un modèle conceptuel de données. La méthode Merise proposée par H. Tardieu et ses collaborateurs en France [TAR 83] comporte une définition précise du formalisme et des règles de construction d'un bon modèle conceptuel. On attribue généralement à l'équipe de Tardieu le choix des termes « entité-association » pour nommer le formalisme, plutôt que « entité-relation », pour éviter toute ambiguïté avec le concept de *relation* provenant de l'algèbre relationnelle, un concept fondamental des bases de données relationnelles mais qui a un tout autre sens dans le formalisme EA.

Le formalisme entité-association fait appel à trois concepts premiers, soit *entité*, *association*, et *attribut* qui sont relatifs à la sémantique des données et trois concepts accessoires, *identifiant occurrence* et *multiplicité*, pour l'expression des contraintes d'intégrité des données du modèle. La modélisation conceptuelle est un processus progressif et descendant où le concepteur s'attarde d'abord à identifier les données importantes, qualifiées de vitales, regroupées sous forme *d'entités*, puis y ajoute les *associations* pertinentes entre ces entités. Sur la base de cette ébauche du modèle, des détails supplémentaires sont ajoutés tels que les données qui doivent être conservées à propos des entités et des associations, appelés les *attributs*, ainsi que d'autres éléments imposant des contraintes sur les données, soit les *identifiants* et les *multiplicités*.

Décrivons maintenant chacun des ces concepts, leur représentation graphique grâce à la notation UML et la sémantique sous-jacente.

Entité, attribut et association

Entité ▸ Objet concret ou abstrait du monde réel au sujet duquel une organisation est susceptible de conserver des données (*Entity*).

Une entité concrète possède une existence physique c'est le cas d'un client, d'un équipement, d'un produit par exemple. Une entité abstraite a une existence conceptuelle, par exemple une transaction, un tarif, l'annulation d'un vol d'avion. Le client **Jean Dupond** est une entité, la commande **COM0001** est aussi une entité, la première concrète, l'autre abstraite. Bien qu'une commande puisse prendre la forme d'un document papier, ce qui intéresse le modélisateur c'est l'aspect conceptuel de la transaction car elle peut ne pas exister sur papier. Elle sera donc considérée avant tout comme un objet abstrait.

Toute entité possède des propriétés, appelés *attributs*, et l'ensemble des entités qui ont les mêmes attributs est représenté graphiquement par une *entité type*, soit un rectangle comportant dans la case au haut le nom de l'entité type et dans la case du bas la liste des attributs de l'entité type. Il s'agit du moule dans lequel toute entité de ce type est formée.

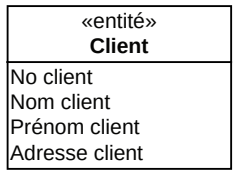
Attribut ▸ Donnée élémentaire qui sert à caractériser une propriété des entités et des associations dans un modèle conceptuel de données (*Attribute*).

La figure 1-1 montre l'entité type **Client** qui décrit la structure commune pour l'ensemble des entités **Client** notamment les attributs que chaque entité de ce type possède : **No client**, **Nom client**, **Prénom client** et **Adresse client**.

Dans le langage UML le rectangle peut représenter divers concepts, par exemple une classe d'objets dans un diagramme de classes. C'est pourquoi il est prescrit par le langage d'ajouter une étiquette au haut du rectangle qui permet d'établir de quel concept il s'agit (Ici le mot « entité »).

En matière de modélisation conceptuelle avec la notation UML, le rectangle ne peut que représenter des entités, c'est pourquoi tout au cours de ce chapitre l'étiquette « entité » sera omise pour éviter d'alourdir les

FIGURE 1-1 **Représentation graphique de l'entité type Client**



modèles. De plus, pour simplifier notre exposé, le terme général *entité* sera utilisé pour parler d'une *entité type* présente dans un modèle conceptuel. On écrira par exemple « L'entité **Client** possède quatre attributs » plutôt que « L'entité type **Client** possède quatre attributs » car cette formulation ne comporte aucune ambiguïté.

Si le terme entité peut représenter l'ensemble des entités de même type, on utilisera fréquemment les termes *occurrences d'une entité* pour faire référence à un élément de cet ensemble.

Occurrence d'entité ▸ Élément particulier d'une entité type, identifiable de façon unique. (*Instance*)

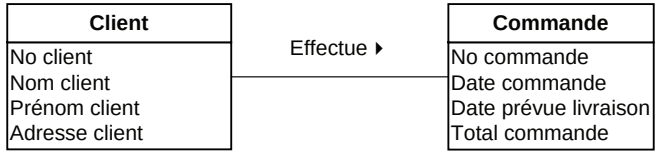
À propos du modèle conceptuel de la figure 1-1 où une entité **Client** est présente, on dira que l'entité dont le **No client** est **CLI0002** est une *occurrence* de cette entité.

Association ▸ Lien sémantique qui existe entre deux entités ou plus. Elle représente souvent la mémoire d'un événement qui a permis d'établir un lien logique entre ces entités (*Relationship*).

Tout comme une entité appartient à une entité type, une association appartient à une *association type* illustrée par un arc entre des entités types dans un modèle conceptuel. En général le terme *association* sera utilisé pour faire référence à une association type dans un modèle conceptuel.

La figure 1-2 présente un modèle où les entités **Client** et **Commande** sont liées par une association portant le nom **Effectue**. On notera que l'association est identifiée à l'aide d'un verbe d'action, qu'une flèche donne une direction à la lecture du modèle (ici de la gauche vers la droite), ce qui permet de faire une lecture de l'association comme s'il s'agissait d'une phrase comportant un sujet, un verbe et un complément. Cette association est dite *binaire* car elle met en cause deux entités. Comme nous le verrons plus loin à la section portant sur les concepts avancés, il existe aussi des associations sur plus de deux entités, dites de *degré supérieur*, qui sont beaucoup moins fréquentes mais dont l'importance mérite que l'on s'y attarde. Nous aurons l'occasion d'y revenir.

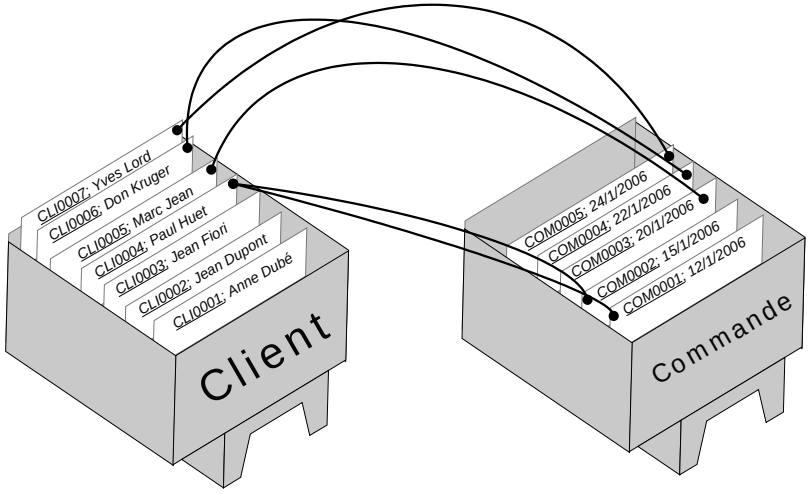
FIGURE 1-2 **Association entre l'entité Client et l'entité Commande**



Le modèle de la figure 1-2 est une représentation abstraite qui, par analogie, pourrait correspondre intuitivement à deux bacs de fiches, le premier portant le nom **Client** et l'autre le nom **Commande**. Certaines fiches du bac **Client** sont reliées à des fiches du bac **Commande** par une ficelle. Dans cette analogie les deux bacs correspondent aux deux entités. Les fiches représentent des occurrences de chacune des entités. Les inscriptions sur les fiches sont les valeurs des attributs pour une occurrence d'entité. Enfin, on assimile les ficelles aux associations. Cette analogie est illustrée à la figure 1-3 où, pour des raisons de simplification, les valeurs pour les attributs des entités ne sont pas toutes montrées. Seules les valeurs des trois premiers attributs le sont : No client, Prénom client et Nom client.

Nous devons à Daniel Pascot de l'Université Laval au Québec l'idée de cette analogie.

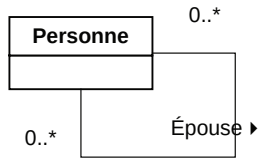
FIGURE 1-3 **Analogie pour le modèle conceptuel de la figure 1-2**



En consultant la figure 1-3, le lecteur comprendra toute l'importance d'une association dans un modèle conceptuel. Sur le plan sémantique, la présence d'une association entre deux occurrences d'entités (une ficelle) nous assure qu'à partir d'une occurrence il est possible d'avoir accès à l'autre occurrence associée en suivant cette ficelle et vice versa.

Une association peut mettre en cause des occurrences de la même entité. On parle alors *d'association réflexive*. Pour illustrer la notion de mariage entre deux personnes, on peut faire appel à une association réflexive comme le montre la figure 1-4. Le modèle illustre le fait qu'un bac comportant des

FIGURE 1-4 **Une personne est associée à une autre personne lorsqu'elle l'épouse**



fiches sur des personnes pourrait comporter des ficelles qui relient des fiches placées dans le même bac pour mémoriser un événement, soit ici le mariage entre deux personnes.

Contraintes sur les attributs et les associations

L'analogie des bacs à fiches met aussi en évidence la nécessité d'identifier chaque fiche dans un bac, donc chaque occurrence d'une entité, pour pouvoir facilement les repérer. Pour ce faire, il doit exister un attribut, ou un groupe d'attributs, qui fournit une identification unique à chaque occurrence d'une entité. Cet attribut ou ce groupe d'attributs est appelé *identifiant* dans le formalisme entité-association. L'identifiant est la clé pour repérer une fiche. Par analogie, l'identifiant pourrait servir à trier les fiches d'un bac et à faciliter le dépistage d'une fiche si la valeur de son identifiant est connue.

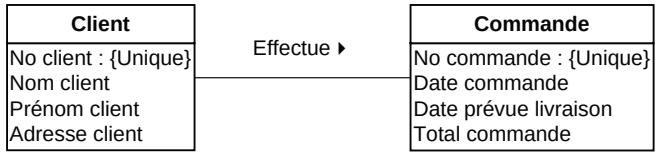
Identifiant ▸ Attribut ou groupe d'attributs permettant d'identifier chaque occurrence d'une entité (*Identifier*).

Pour ce qui est du modèle conceptuel de la figure 1-2, on peut déduire sans risque de se tromper que l'attribut **No client** serait un bon choix comme identifiant de l'entité **Client** et que le **No commande** pourrait jouer ce rôle pour l'entité **Commande**. On en conclut que chaque numéro de client et chaque numéro de commande est unique, ce qui permet de repérer une fiche dans le bac correspondant si son identifiant est connu. L'importance de l'identifiant implique que chaque fiche possède une valeur pour l'identifiant et que les fiches soient classées selon l'ordre croissant de la valeur de l'identifiant.

La plupart des notations utilisent le soulignement pour marquer un identifiant. Dans la notation UML cependant, il en va autrement. L'attribut est marqué d'une *contrainte*, placée à la suite du nom de l'attribut, qui indique que la valeur de cet attribut est *unique*, c'est-à-dire qu'elle ne peut être prise par deux occurrences de la même entité. En UML une contrainte est inscrite entre accolades comme le montre la figure 1-5. Ici la contrainte {Unique} marque l'attribut agissant comme identifiant.

Il s'agit d'un premier type de contrainte applicable à un attribut qui doit OBLIGATOIREMENT avoir une valeur pour chaque occurrence de l'entité. Par ailleurs, les valeurs doivent être toutes différentes d'une occurrence à l'autre. La présence d'une contrainte {Unique} sur un attribut a pour corollaire que la valeur de cet attribut ne peut être inconnu, ou nul. Cette contrainte n'est pas expressément exprimée dans le modèle sous la forme {Unique ; Non nul} mais elle est implicite pour l'attribut choisi comme identifiant. La présence d'un identifiant marqué de la contrainte {Unique} pour une entité représente une *contrainte d'intégrité d'entité*.

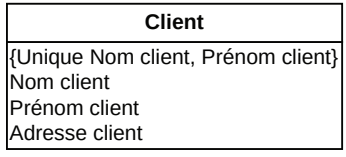
FIGURE 1-5 **Modèle conceptuel dont les identifiants sont marqués**



Lorsqu'un identifiant est formé d'un seul attribut on le qualifie d'identifiant *simple*. S'il s'avère qu'un identifiant doive comporter plusieurs attributs, une contrainte {Unique} sera placée avant tous les attributs de l'entité indiquant en cela quels sont les attributs qui forment l'identifiant. On parle alors d'un identifiant *composé*.

La figure 1-6 montre une entité avec identifiant composé. Comme **No client** n'est pas présent dans l'entité **Client**, la combinaison du **Nom** et du **Prénom** du client a été choisi comme identifiant dans la mesure où on peut être assuré que deux clients n'auraient jamais à la fois le même nom ET le même prénom. Si on ne peut être assuré de cela, il vaut mieux envisager un nouvel attribut, **No client**, comme le montre l'entité **Client** de la figure 1-5.

FIGURE 1-6 **Modèle conceptuel avec identifiant composé**



Bien qu'il ne s'agisse pas d'une règle absolue, il est peu recommandé d'avoir des identifiants composés dans un modèle conceptuel s'il est destiné à être traduit en un modèle logique. Nous verrons au chapitre 2 que la présence d'identifiants composés rend inutilement complexe un modèle logique. Si le modélisateur ne peut trouver pour une entité un attribut naturel qui puisse convenir comme identifiant, il aura toujours le loisir de créer un attribut artificiel qui va jouer ce rôle. Ainsi, par exemple, dans une organisation qui reçoit des c.v. pour les postes affichés, même si les c.v. sont conservés traditionnellement dans un dossier pour chacun des postes, et ce, en ordre alphabétique selon le nom des candidats, le modélisateur peut facilement convenir avec les utilisateurs du système à développer que l'on utilisera un attribut artificiel tel que **No CV** ou **No candidat** pour identifier chaque c.v. reçu.

1-1 Règle d'identité

Toute entité présente dans un modèle conceptuel de données doit comporter **obligatoirement** un identifiant. Chaque occurrence de l'entité doit posséder une valeur pour cet attribut. La valeur de l'attribut identifiant devra être **stable**, c'est-à-dire qu'au cours de la vie d'une occurrence de l'entité cette valeur ne pourra changer. De plus deux occurrences de l'entité ne pourraient avoir la même valeur pour leur identifiant. La même valeur accordée plus d'une fois à un identifiant représente un **doublon**. Les doublons sont inacceptables pour un identifiant.

La règle d'identité prescrit qu'un identifiant est obligatoire, stable et sans doublon. Dans le cas de l'entité **Client**, au delà du fait que le choix d'un identifiant composé n'est pas recommandé, l'utilisation du nom et du prénom du client ne peut garantir la stabilité de l'identifiant, car une personne pourrait changer de nom. De plus comme nous l'avons évoqué plus tôt, la présence de doublons est fort probable à moins que l'on ne décide d'écarter tout client potentiel qui soit l'homonyme d'un client existant!!!

Un autre type de contrainte doit être ajouté au modèle conceptuel dès lors que les identifiants des entités associées sont établis. Il s'agit des *contraintes de multiplicité* qui précisent à la fois que la participation d'une entité à l'association est obligatoire ou non, et le cas échéant, le nombre d'occurrences d'une association auxquelles elle peut participer.

Multiplicité (association binaire) ▸ Contrainte inscrite à chaque extrémité d'une association binaire comportant un couple de valeurs (minimum–maximum) qui établit, pour chaque entité de l'association, les nombres minimum et maximum d'occurrences de l'autre entité qui peuvent lui être associées (*Multiplicity*).

Les contraintes de multiplicité dépendent fortement des règles d'affaires (*business rules*) d'une organisation. Il est fondamental que le modélisateur et l'ensemble des utilisateurs du système aient une compréhension commune de ces règles pour que les contraintes de multiplicité soient valables et s'avèrent en conséquence pertinentes.

Le tableau 1-1 donne les quatre combinaisons de base du couple de valeurs. Le premier chiffre (0 ou 1), indique qu'il s'agit d'une association *optionnelle* (0) ou d'une association *obligatoire* (1). Le deuxième chiffre indique le nombre maximum d'occurrences des associations auxquelles une entité participe : *une* (1) ou strictement plus d'une, c'est-à-dire *plusieurs* (*). L'astérisque représente en effet la valeur *plusieurs*.

TABLEAU 1-1 **Codification des multiplicités**

Multiplicité UML	Signification
`0..1`	Au plus un
`1..1 (ou 1)`	Un seul
`0..* (ou *)`	Un nombre indéterminé
`1..*`	Au moins un

Si nous reprenons le modèle de la figure 1-4 et tentons d'établir les multiplicités de l'association binaire **Effectue**, il suffit de consulter l'analogie de la figure 1-3 pour arriver à les déduire.

Considérons les occurrences de **Client**, soit par analogie les fiches dans le bac **Client**. Nous notons que certains clients ne sont liés à aucune commande, c'est le cas d'Anne Dubé. L'association avec **Commande** est donc optionnelle (0) pour **Client**. Par ailleurs, certains clients sont liés à plusieurs commandes (*), c'est le cas de Paul Huet. On en déduit donc la multiplicité 0..* qui sera placée sur l'association du côté de **Commande**.

En consultant les occurrences de **Commande**, soit les fiches du bac **Commande**, on constate qu'elles sont toutes liées à un client. L'association avec **Client** est donc obligatoire (1). De plus une commande n'est jamais liée à plus d'un client (1). On en déduit la multiplicité 1..1 qui sera placée sur l'association du côté de **Client**, car une commande ne peut être liée qu'à un seul **Client**.

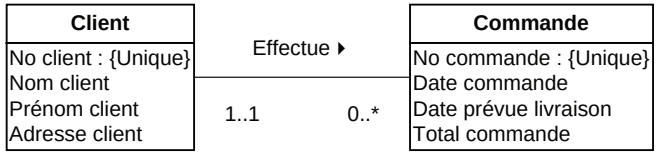
Le modélisateur expérimenté ne fait pas nécessairement la réflexion sur les multiplicités à l'aide de l'analogie du bac. Il va établir assez aisément, en analysant le fonctionnement de l'organisation, qu'un client peut effectuer

un nombre indéterminé de commandes (0..*) et qu'une commande ne peut être liée qu'à un seul client (1..1). En toute logique, une commande ne peut en effet provenir de deux clients.

La figure 1-7 montre un modèle conceptuel maintenant complet avec les deux contraintes de multiplicité placées de part et d'autre de l'association. Toute association binaire comporte deux multiplicités car une association se lit dans les deux directions avec des multiplicités qui peuvent être différentes. Dans le modèle de la figure 1-7 le modélisateur a été tenu de répondre à deux questions qui ont conduit à deux réponses différentes :

1. « Un client est-il obligatoirement lié à une commande et à combien au maximum ? »
2. « Une commande est-elle obligatoirement liée à un client et à combien au maximum ? »

FIGURE 1-7 **Modèle conceptuel avec multiplicités**



Contraintes de domaine des attributs

Un modèle conceptuel, dans sa première version, doit présenter les entités avec leur identifiant et leurs autres attributs, ainsi que les associations avec leurs multiplicités. D'autres contraintes peuvent par ailleurs apparaître sur le modèle dans sa version finale pour assurer la transition vers le modèle logique. Ces contraintes, appelées *contraintes de domaine de l'attribut,* vont fixer le type de données de l'attribut et les valeurs admissibles de l'attribut.

Tous les outils de modélisation permettent d'inscrire ces contraintes dans ce qui est souvent appelé le *dictionnaire de données* du modèle. En revanche, elles ne sont pas toujours affichées dans les entités pour éviter d'alourdir le modèle.

Lorsqu'elles sont affichées, elles apparaissent à la suite du nom de l'attribut, précédées du deux-points (**:**). Leur formulation peut être libre ou donnée grâce à un langage spécialisé tel que OCL (*Object Constraint Language*).

Les types de données de base proposés par le langage OCL sont donnés dans le tableau 1-2.

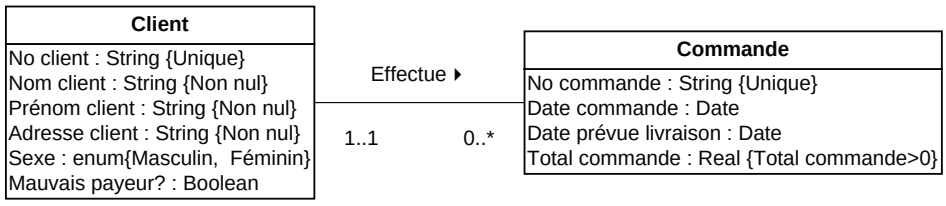
TABLEAU 1-2 **Types de données pour spécifier le domaine des attributs**

Type	Signification
`Integer`	Valeur numérique entière
`Real`	Valeur numérique réelle
`String`	Chaîne de caractères
`Date`	Date du calendrier
`Boolean`	Valeur Vrai ou Faux
`enum{}`	Liste de valeurs admissibles

On y retrouve les types de base pris en charge par la plupart des SGBD. Le type `enum` est utilisé pour énumérer, s'il y a lieu, les valeurs admissibles du domaine de l'attribut. Le domaine d'un attribut peut en effet être contraint à la fois à un type de données et à une liste de valeurs admissibles de ce type. Nous avons placé dans le tableau le type `Date` à cause de son importance dans les bases de données bien qu'il ne s'agisse point d'un type prescrit par OCL. En effet, sur le plan technique une date est un cas particulier du type `Real`.

S'il y a lieu, le type de données sera suivi d'une contrainte indiquant une restriction sur les valeurs que l'attribut pourra prendre. Cette contrainte est inscrite entre accolades et formulée selon les exigences du langage OCL. La figure 1-8 reprend le modèle conceptuel de la figure 1-7 où deux attributs ont été ajoutés à l'entité **Client** afin d'illustrer le type `enum` et le type `Boolean`. Chaque attribut possède maintenant son type de données. Les attributs **Nom client**, **Prénom Client**, **Adresse Client**, **Total commande** possèdent par ailleurs une contrainte sur la valeur de l'attribut. En vertu de ces contraintes, tout client doit avoir à la fois un nom, un prénom et une adresse. Le sexe du client, s'il est connu est déterminé par les mots *Masculin* ou *Féminin*. Le client peut être ou non mauvais payeur. Le total d'une commande doit être strictement supérieur à zéro.

FIGURE 1-8 **Modèle conceptuel avec contraintes de domaine des attributs**



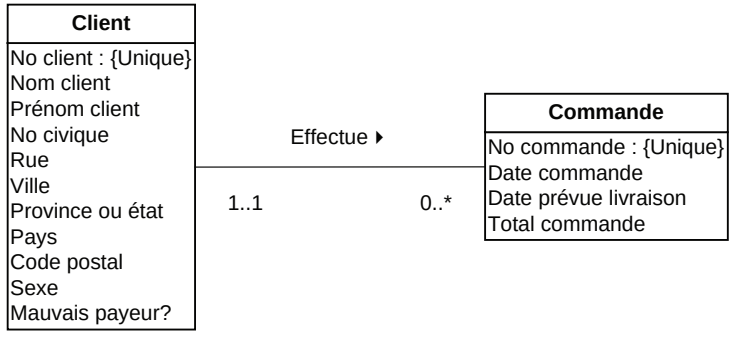
Pour le reste de ce chapitre, nous allons omettre de montrer dans nos modèles conceptuels les contraintes de domaine des attributs. Cela n'a pas pour but de réduire leur importance car ces contraintes appartiennent de plein droit au niveau conceptuel, au même titre que les identifiants et les associations. Elles ne sont cependant pas essentielles. Si le modélisateur fait appel à un outil de modélisation, elles ne sont généralement pas affichées dans les entités afin d'alléger la présentation puisqu'elles peuvent être consultées dans le dictionnaire de données du modèle. Le lecteur comprendra qu'elles sont en général sous-jacentes aux modèles conceptuels présentés.

Tous les types de données du tableau 1-2 sont des types simples, c'est-à-dire que l'attribut qui en est doté ne peut prendre qu'une seule valeur dans une occurrence. Le sexe du client est Masculin ou Féminin, jamais les deux à la fois. **Mauvais payeur ?** a une seule valeur soit Vrai, soit Faux. Notons au passage qu'un attribut de type booléen porte souvent un nom suivi du point d'interrogation. Il ne s'agit pas d'une règle stricte de modélisation conceptuelle mais une simple convention permettant de les repérer facilement.

La valeur de l'attribut **Adresse client** est UNE SEULE suite de caractères, incluant les espaces, ne comportant aucune structure imposée par le type de données `String`. Si le modélisateur veut imposer une structure à l'adresse du client il devra identifier autant d'attributs que nécessaire pour définir les composants de l'adresse, par exemple **No civique**, **Rue**, **Ville**, **Province ou état**, **Pays**, **Code postal**. L'attribut **Adresse client** devient alors inutile.

La figure 1-9 montre un modèle conceptuel où la structure de l'adresse est définie explicitement.

FIGURE 1-9 **Modèle conceptuel avec des attributs structurant l'adresse**



La nécessité de faire appel à un type de données simple pour chaque attribut du modèle nous conduit à proposer une nouvelle règle incontournable en matière de modélisation conceptuelle des données. Cette règle impose que dans une occurrence chaque attribut ne peut avoir simultanément plusieurs valeurs. La valeur associée à un attribut d'une occurrence sera donc en toute circonstance une donnée élémentaire non décomposable et unique.

1-2 Règle de description

Chaque attribut d'une entité présente dans le modèle conceptuel ne peut prendre qu'une seule valeur à la fois pour son occurrence. Cette valeur doit provenir du domaine décrit par les contraintes de domaine de l'attribut.

Dépendance fonctionnelle des attributs à l'identifiant

Une grande difficulté de la modélisation conceptuelle des données, surtout pour le modélisateur novice, est de faire en sorte que chaque attribut dans une entité représente une propriété qui n'appartient strictement qu'à cette entité.

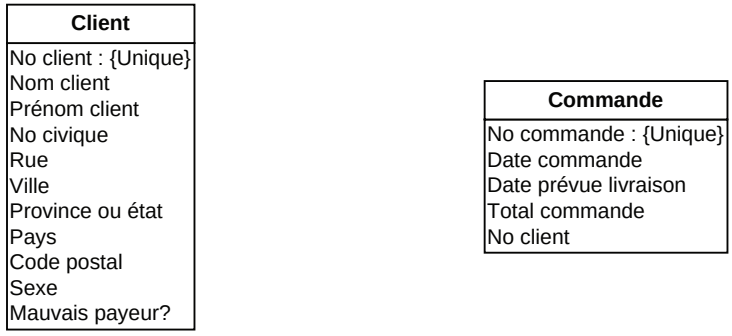
Pour guider le modélisateur dans sa tâche, deux nouvelles règles sont proposées.

1-3 Règle de non redondance

Chaque attribut du modèle conceptuel est unique, il ne peut y apparaître plus d'une fois à travers plusieurs entités du modèle.

La figure 1-10 illustre une erreur fréquente commise par le modélisateur novice. Un attribut est créé dans une entité qui ne représente pas une propriété de l'entité. **No client** est une propriété qui appartient strictement à

FIGURE 1-10 **Modèle conceptuel avec un attribut redondant**



l'entité **Client** et ne peut apparaître dans une autre entité. Le modélisateur a probablement voulu souligner qu'il y a un lien entre une commande et un client. Il eut mieux valu qu'il crée une association pour mettre en évidence ce fait, comme le montre la figure 1-9, plutôt que de créer un attribut redondant, ce qui transgresse la règle 1-3.

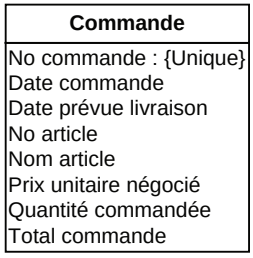
1-4 Règle de construction

Une entité ne peut en contenir une autre. En particulier, une entité ne peut contenir un attribut d'une autre entité.

La figure 1-11 montre une entité **Commande** comportant des attributs qui décrivent une autre entité. En effet, **No article** et **Nom article** sont visiblement des attributs d'une autre entité qui a une existence propre : **Article**. De plus, **Prix unitaire négocié** et **Quantité commandée** relèvent manifestement davantage de l'article que de la commande.

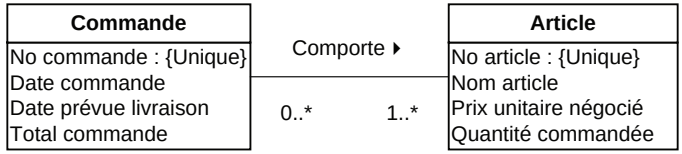
La règle de construction est ici transgressée, car deux entités types sont assimilées à une seule. Par ailleurs, en vertu de la règle de description, le modèle nous indique qu'une commande ne peut comporter qu'un seul article. La règle de description le prescrit formellement : pour une occurrence de **Commande**, **No article** ne doit avoir qu'une seule valeur. L'entité **Commande** est non seulement mal construite, mais elle est invalide, car elle ne reflète pas la réalité : une commande pourrait comporter plus d'un article.

FIGURE 1-11 **Modèle conceptuel transgressant la règle de construction**



Une première version d'un modèle conceptuel permettant de respecter les règles de description et de construction est donnée à la figure 1-12. Il montre une séparation nette entre les attributs qui décrivent une commande et les attributs qui décrivent un article avec une association qui établit un lien entre ces entités. Les multiplicités montrent qu'une commande comporte au moins un article et qu'un article peut apparaître dans un nombre indéterminé de commandes.

FIGURE 1-12 **Modèle conceptuel révisé, première version**



Une notion théorique importante est à la base de la règle de construction : *la dépendance fonctionnelle*. Pour que la règle de construction soit respectée, il faut que chaque attribut autre que l'identifiant *dépende fonctionnellement* de l'identifiant. On se souviendra qu'une fonction mathématique assure qu'un élément dans un ensemble mène à un et un seul élément dans un autre ensemble. L'élément du deuxième ensemble est dit dépendant fonctionnellement de l'élément du premier ensemble. Dans le contexte d'un modèle conceptuel, tout attribut autre que l'identifiant doit dépendre fonctionnellement de l'identifiant intégralement, même lorsque l'identifiant est composé. Exprimé différemment, la règle de construction stipule qu'une valeur de l'identifiant détermine une et une seule valeur pour chaque attribut autre que l'identifiant.

On peut être assuré qu'une commande, la commande avec l'identifiant **COM0001** par exemple, mène à (ou détermine) une seule valeur pour **Date commande**, une **Date prévue livraison** et un seul **Total commande**. C'est bien le cas et on en conclut que ces trois attributs dépendent tous fonctionnellement de l'identifiant. Ils peuvent donc être considérés comme des attributs propres à l'entité **Commande**. La règle de construction est dès lors satisfaite.

Quant à l'entité **Article**, il va de soi que **No article** détermine la valeur de **Nom article**. Connaissant le numéro de l'article, le nom de l'article est identifiable et unique.

On peut en revanche douter que le numéro d'un article puisse déterminer la quantité commandée. En effet, un article peut être commandé en quantité différente d'une commande à l'autre. Il en va de même pour **Prix unitaire négocié** qui n'est probablement pas systématiquement le même pour un article donné, par exemple une boîte de papier mouchoir, car s'il est négocié avec le client il pourrait varier d'une commande à l'autre, selon le client. On en conclut que **No article** ne détermine ni **Prix unitaire négocié**, ni **Quantité commandée**. En fait ces deux attributs dépendent fonctionnellement de l'article ET de la commande. Un attribut qui dépend fonctionnellement de deux entités ou plus doit être considéré comme un *attribut de l'association*.

Les attributs d'une association sont représentés en UML à l'aide d'une entité rattachée à l'association par une ligne pointillée. On pourrait qualifier cette entité *d'entité d'association*. Il peut être nécessaire de définir des entités d'association pour regrouper des attributs qui ne peuvent appartenir ni à l'une ni à l'autre des entités associées et éviter ainsi de transgresser la règle de construction. Bien qu'il s'agisse sur le plan sémantique d'une entité, son identifiant n'y est généralement pas inscrit car il résulte implicitement de la combinaison des identifiants des entités associées. Une entité d'association est en fait ce que les modélisateurs appellent une *entité faible*.

Entité faible ▸ Type d'entité dont l'existence dépend de deux ou plusieurs entités. Son identifiant se définit en fonction des identifiants des entités dont elle dépend. Elle ne possède pas d'attribut qui puisse servir d'identifiant et qui lui appartienne en propre.

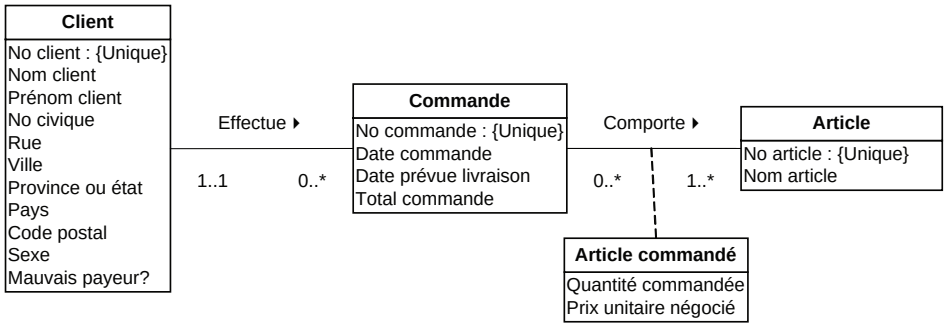
Une entité faible est définie par opposition aux entités dites *fortes*, entités qui possèdent un ou des attributs pouvant constituer un bon identifiant.

Une entité d'association doit respecter toutes les règles de modélisation conceptuelles présentées plus tôt, incluant la règle d'identité. Le modélisateur doit prendre garde de créer une entité d'association non justifiée. Ce risque est grand lorsque les multiplicités maximales de l'association sont toutes les deux plusieurs (*). Si la combinaison des identifiants des entités associées ne détermine pas chaque attribut de l'entité d'association, le modélisateur devra se rendre à l'évidence que les attributs de l'association sont en fait les attributs d'une entité forte et qu'il ne peut s'agir d'une association avec attributs. Nous aurons le loisir dans le cadre de l'étude de cas 1-3 traitée plus loin de montrer comment une entité d'association peut être un mauvais choix dans certains cas.

La figure 1-13 montre un modèle conceptuel qui respecte à la fois les règles de description et de construction grâce à l'emploi d'une entité d'association.

Le choix d'un nom qui soit pertinent pour une entité d'association n'est pas une tâche facile. Il doit permettre de représenter fidèlement le concept qui donne lieu à des attributs propres à l'association. Le modèle de la figure 1-13 permet de voir que le modélisateur a clairement distingué le concept **Article** d'une part, pour faire référence à l'article du catalogue, et d'autre part l'article présent dans une commande soit **Article commandé**. Son identifiant est ici implicite, il est composé de **No commande** et **No article**. Il est valide car un numéro d'article donné, par exemple l'article **10007**, n'apparaîtra probablement qu'une fois dans une commande. La combinaison **No commande-No**

FIGURE 1-13 **Modèle conceptuel final avec une entité d'association : Article commandé**



ARTICLE mène en effet à une seule occurrence de **Article commandé**. Celle-ci est dès lors considérée comme une entité faible et le modélisateur a donc fait un bon choix en la considérant comme une entité d'association.

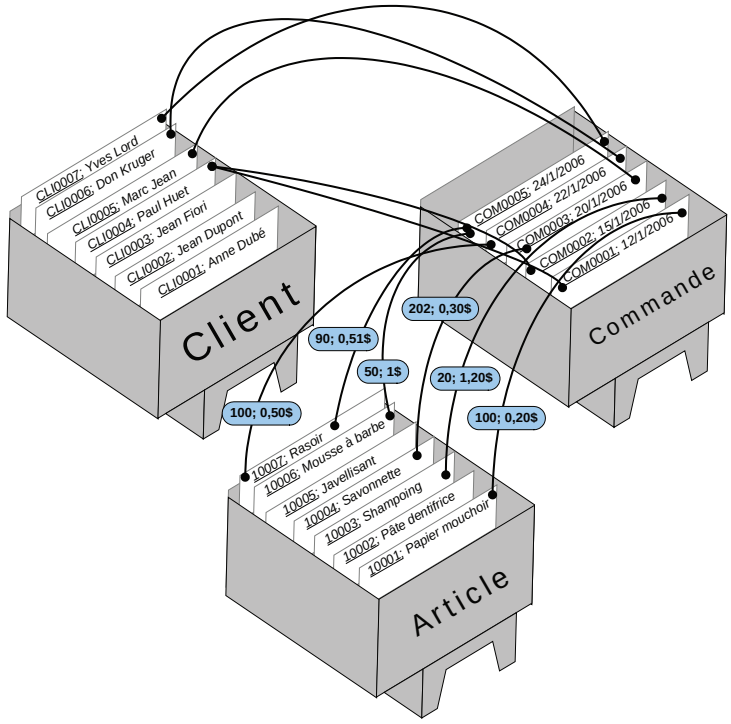
Revenons à l'analogie du bac. Une occurrence d'une entité d'association peut être considérée comme une étiquette attachée à la ficelle tenant lieu d'association. Cette étiquette contient les valeurs des attributs de l'entité d'association et elle est indissociable de la ficelle qui la retient.

On peut voir à la figure 1-14 une représentation visuelle du modèle conceptuel de la figure 1-13. Le lecteur comprendra à l'aide de cette représentation la distinction que le modélisateur a voulu faire sur le plan sémantique entre une commande, un article et un article commandé. Tous ces concepts sont évidemment associés mais aucun ne véhicule de données redondantes.

Les ficelles qui relient une fiche du bac **Commande** à une fiche du bac **Article** possèdent toutes une étiquette où apparaissent deux valeurs : la quantité de l'article commandé et son prix unitaire négocié. C'est bien ce que reflète le modèle conceptuel avec l'entité d'association **Article commandé** et ses deux attributs. Chaque attribut de l'étiquette n'a qu'une seule valeur (règle de description). Chaque étiquette a une identité donnée par la combinaison du numéro d'article et du numéro de commande (règle d'identité). Les valeurs des étiquettes sont déterminées par l'identifiant (règle de construction).

Considérons par exemple la combinaison **COM0005-10007** à titre de valeur de l'identifiant pour une occurrence de l'entité d'association **Article commandé**. Il ne réfère qu'à une seule occurrence de cette dernière qui a par ailleurs la seule et unique valeur **90** pour **Quantité commandée**.

FIGURE 1-14 **Analogie pour le modèle conceptuel de la figure 1-13**



Les règles fondamentales proposées dans cette section pour réaliser des modèles conceptuels sémantiquement corrects à l'aide du formalisme entité-association n'assurent aucunement que les modèles produits soient valides.

Un modèle conceptuel est valide s'il reflète une réalité et répond de manière adéquate aux besoins en information. Comme un modèle est une simplification de la réalité, il n'est jamais qu'un reflet de cette réalité. Cette réalité est perçue selon une perspective particulière (nous dirions un point de vue), quelquefois biaisée, celle du modélisateur.

La section qui suit présente quelques cas simples de modélisation illustrant les concepts et les règles énoncés plus haut. Elle est suivie d'un exposé de lignes directrices permettant au modélisateur de faire un choix adéquat des entités, des associations et de leurs attributs, dans le cadre particulier de l'analyse des besoins préalables à la réalisation d'un système d'information.

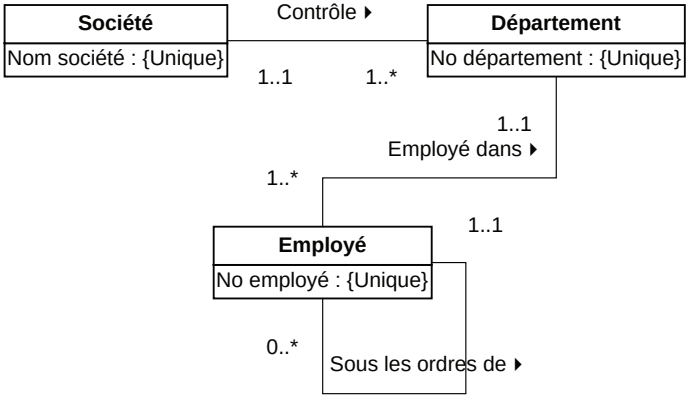
Il est recommandé au lecteur de tenter de produire son propre modèle conceptuel à partir de la description de chaque cas avant de consulter le modèle proposé à titre de solution par l'auteur.

CAS SIMPLES DE MODÉLISATION CONCEPTUELLE DES DONNÉES

CAS 1-1 SOCIÉTÉ ET DÉPARTEMENT

Créer avec le formalisme entité-association un modèle conceptuel de données qui illustre la situation suivante :

1) Chaque société contrôle des départements appartenant à une seule société ;
2) Chaque département emploie un ou plusieurs employés et chaque employé travaille pour un seul département ;
3) Un employé peut avoir aucun ou plusieurs subalternes et chaque subalterne est sous les ordres d'un seul employé.



Le modélisateur a fait appel à une association réflexive, **Sous les ordres de**, pour traduire la notion de subalterne. Comme on le constate, une notion ou un concept ne se traduit pas automatiquement en entité. Le lecteur notera que toutes les associations sont obligatoires (les multiplicités minimales ont la valeur 1) sauf une : l'association **Sous les ordres de** est obligatoire dans une direction et pas dans l'autre. En effet dans une direction on lit qu'un employé est obligatoirement sous les ordres d'un autre employé et un seul. Dans l'autre direction un employé peut ne pas être associé à des subalternes.

Le modèle est bien construit. Chaque entité possède au moins un attribut, chaque entité possède son identifiant et chaque association binaire possède deux multiplicités. De plus les contraintes de multiplicité stipulent les règles d'affaires suivantes conformes aux exigences du cas :

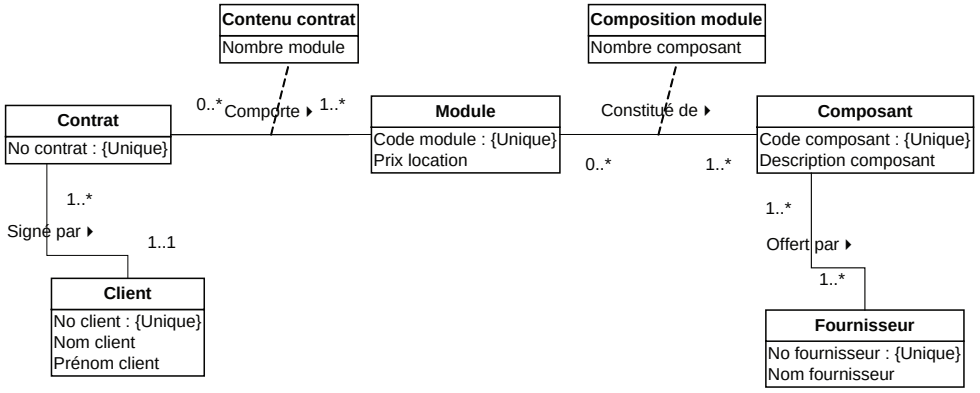
- une société contrôle au moins un département

- un département relève d'une seule société
- un employé est employé dans un seul département
- un département emploie au moins un employé
- un employé est sous les ordres d'un seul employé
- un employé dirige un nombre indéterminé d'employés (0 ou plusieurs)

CAS 1-2 LOCATEUR D'ABRIS

(DP) Créer avec le formalisme entité-association un modèle conceptuel de données qui décrit les exigences de données pour cette entreprise :

1) Une entreprise de location d'abris pour automobiles propose à ses clients des abris modulaires. Le client signe un contrat pour la location d'un ou plusieurs modules de divers types qui peuvent être combinés pour répondre à ses besoins spécifiques.
2) Tout client possède un numéro de client, un nom et un prénom. Un contrat peut comporter des modules de types différents et un nombre indéterminé de modules du même type.
3) Chaque type de module possède un code et un prix de location propre. Il est constitué de divers composants standards. Le même type de composant peut être présent en plusieurs exemplaires dans un même module et dans plusieurs modules différents.
4) Le composant possède un code de composant et une description. Un composant pourrait être un boulon de 2 cm par exemple.
5) L'entreprise se procure les composants auprès d'un certain nombre de fournisseurs attitrés. Un fournisseur possède un numéro unique et un nom.



Le modèle reflète la structure des modules : ils sont constitués d'un certain nombre de composants standards obtenus de divers fournisseurs. Le **Nombre de composant** est déterminé à la fois par le **Code composant** et le **Code module**. Ce ne peut être qu'un attribut de l'association **Constitué de** qui permet de rattacher un type de module à ses divers composants. L'entité d'association **Composition module** est une entité faible car l'identifiant implicite combinant **Code module-Code composant** est valide : un code module donné et un code composant donné détermine un et un seul nombre de composants. Chaque contrat possède des types de modules dont le nombre, pour un module donné, varie d'un contrat à l'autre. **Nombre module** ne peut être qu'un attribut de l'association liant le contrat

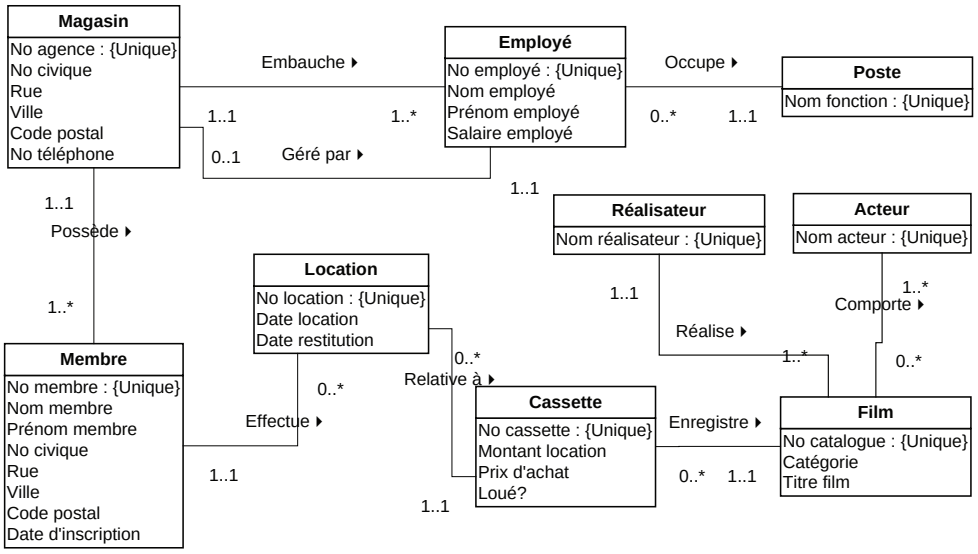
aux types de modules requis par le client. **Contenu contrat** est aussi une entité faible. On notera enfin les multiplicités minimales à 0 pour la lecture de droite à gauche des associations **Comporte** et **Constitué de**. En effet, un type de module peut exister sans ne jamais avoir fait l'objet d'un contrat et de la même manière, un type de composant peut exister sans qu'il n'ait encore été utilisé dans un module.

CAS 1-3 VIDÉO CLUB

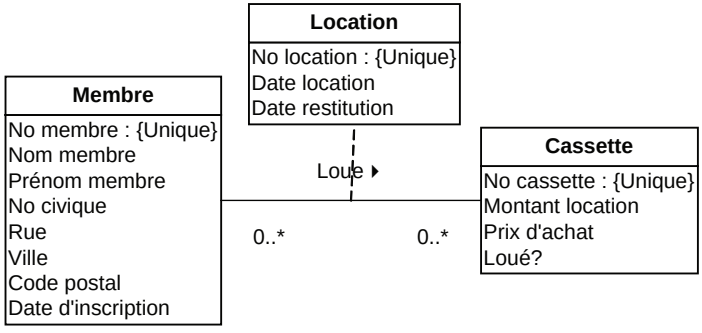
(DP) Créer avec le formalisme entité-association un modèle conceptuel de données qui décrit les exigences de données pour cette société :

1) Une société de location de cassettes vidéo compte plusieurs magasins dans tout le pays. Chaque magasin possède une adresse constituée du numéro civique, de la rue, de la ville, du code postal et d'un numéro de téléphone. Le magasin reçoit un numéro d'agence, unique pour l'ensemble de la société.
2) Chaque magasin embauche son propre personnel, dont un gérant. Les données qui concernent un employé sont son nom, son prénom, son poste et son salaire. Chaque employé reçoit un numéro unique pour l'ensemble de la société.
3) Chaque magasin possède un stock de cassettes vidéo. Les données mémorisées à propos d'une cassette sont le numéro de la cassette, le numéro de catalogue du film, le titre du film, sa catégorie, le montant de location quotidien, le prix d'achat de la cassette, l'état (loué ou non), les noms des principaux acteurs et du réalisateur du film. Le numéro de catalogue identifie chaque film, mais puisqu'il existe plusieurs copies du même film, le numéro de la cassette est unique à chaque cassette.
4) Le film fait partie d'une catégorie parmi les suivantes : action, famille, drame, comédie, horreur, fantastique, science-fiction ou adulte. L'état indique si la cassette est disponible en location.
5) Avant de pouvoir louer une cassette dans un magasin de location, un client doit s'inscrire comme membre auprès d'un magasin de location. Les données mémorisées concernant le client sont son nom et son prénom, son adresse et la date d'inscription du membre. Chaque membre reçoit un numéro de membre unique parmi tous les magasins de location de la société.
6) Les données sur chaque cassette louée sont un numéro de location, le nom du membre, le numéro de la cassette, le titre du film, les dates de location et de restitution. Le numéro de location est unique pour l'ensemble de la société.

Le modèle montre deux associations entre **Magasin** et **Employé**, une première permettant de rattacher l'employé à un magasin et une deuxième permettant d'établir quel employé agit comme gérant d'un magasin. **Poste** est un concept qui a son existence propre, ce qui justifie d'en faire une entité de plein droit même si elle ne possède qu'un seul attribut. **Cassette** et **Film** sont nettement deux entités, d'autant qu'un film pourrait être enregistré sur plusieurs cassettes. Une location existe dans le contexte de l'emprunt d'une cassette par un membre, ses attributs dépendent à la fois de **Membre** et de **Cassette**.



Le modélisateur aurait pu tomber dans le piège faisant de **Location** une entité d'association sur la simple base que **Date location** ou **Date restitution** dépendent à la fois de **No membre** et **No cassette**. Mais il ne s'agit pas d'une dépendance fonctionnelle. En effet un numéro de membre combiné à un numéro de cassette peut mener à plusieurs dates de location puisqu'un membre pourrait théoriquement louer la même cassette plus d'une fois. **Location** est une entité forte, elle possède son propre identifiant : **No location**. L'utilisation d'une entité d'association telle qu'illustrée ci-après est doublement INJUSTIFIÉE pour lier **Membre** à **Cassette**. Le modélisateur a évité ce piège.



Le modélisateur a judicieusement donné l'attribut **Catégorie** au film, non pas à la cassette. Il s'agit strictement d'un attribut du film. Le type de données de **Catégorie** aurait pu être inscrit à la suite du nom de l'attribut sous la forme **enum{action, famille, drame, comédie, horreur, fantastique, science-fiction, adulte}**, mais le modélisateur a choisi de ne pas le faire apparaître sachant qu'il pourra le faire par le biais du dictionnaire de données du modèle.

CHOISIR LES ENTITÉS, LES ASSOCIATIONS ET LES ATTRIBUTS

Le modèle conceptuel de données est une représentation visuelle des entités dont les attributs établissent quelles sont les données dont il faudra assurer éventuellement la persistance dans une base de données. Puisqu'une base de données met en œuvre une variété et une quantité finie de données persistantes, le modélisateur devra faire une sélection parmi les données qui feront l'objet d'une représentation visuelle dans un modèle. Pour ce faire, il sera guidé par un certain nombre de principes.

Étant donné que le système d'information qui doit être réalisé aura à répondre aux besoins en information d'une organisation ou d'un secteur cible de l'organisation, le travail premier du modélisateur est de déterminer quelles sont les données qui sont *vitales* au fonctionnement du secteur visé.

Nous aurons l'occasion au chapitre 4 de proposer une démarche d'analyse et de conception de bases de données qui s'appuie sur la notion de donnée vitale. Pour l'instant nous nous en tiendrons à évoquer et illustrer ces principes qui s'avèrent utiles dans la phase d'analyse des besoins en information.

Principes suggérés

Comment choisir les données à modéliser ?

Principe **REPÉRER LES DONNÉES VITALES.** Une stratégie qui a fait ses preuves pour identifier les données à modéliser repose sur la notion de donnée vitale. Une donnée vitale est une donnée sans laquelle une organisation ne pourrait gérer ses opérations correctement. Cela signifie que si l'organisation ne dispose pas de cette donnée, sa mission et l'atteinte de ses objectifs pourraient être compromises.

L'identification des données vitales doit se faire par le modélisateur avec le concours des utilisateurs du système d'information. Cette démarche doit permettre de distinguer les données essentielles (celles dites vitales) et les données accessoires dans l'organisation. Le travail de modélisation devrait débuter avec le repérage des données vitales.

Une même donnée peut être jugée vitale dans une organisation et non vitale par une autre. Par exemple si toutes les organisations devraient conserver des données sur leurs employés, il n'est pas nécessairement vital de conserver la taille des employés dans un système d'information, à moins qu'il ne s'agisse d'une agence de mannequins pour laquelle cette donnée est en effet indispensable.

On considère qu'une donnée pouvant être dérivée d'une autre donnée, ou d'un ensemble de données, n'est pas vitale. Par exemple, si nous disposons de la date de naissance de l'employé, il n'est pas vital d'avoir aussi son âge, ce dernier pouvant être dérivé de la date de naissance si la date de naissance est jugée vitale. Dans d'autres circonstances, une donnée dérivée peut être vitale. Ainsi le total d'une commande, bien qu'il puisse être dérivé du nombre d'articles, de leur prix respectif et des taxes, sera généralement considéré vital pour des raisons contractuelles. Le total de la commande calculé au moment de la commande est celui qui devra être payé même si, quelques jours après la transaction, le prix d'un article de la commande devait être majoré de 10% dans le catalogue du fournisseur.

Principe **Repérer les données vitales dans les documents existants.** Une source importante de données vitales provient du système d'information d'une organisation (ses formulaires, ses bordereaux, ses procédures et ses ressources) et, s'il est informatisé, des fichiers ou des bases de données du système.

À travers les nombreux exemples ainsi que les cas de modélisation proposés dans cet ouvrage, il sera fait couramment usage de cette stratégie pour illustrer divers aspects de la modélisation conceptuelle. C'est notamment le cas dans la dernière partie de cette section où des études de cas sont menées à partir de documents et de formulaires utilisés par un système d'information existant.

Comment faire la différence entre un attribut et une entité ?

Une entité représente généralement un groupe de données vitales. Un attribut représente une donnée vitale de type simple ou élémentaire : un chiffre (entier ou réel), une chaîne de caractères, une date, un booléen.

Principe **Distinguer entité et attribut sur la base de leur structure.** Un attribut ne peut prendre qu'une valeur élémentaire. Une entité aura une structure complexe, soit au moins un attribut, même si initialement le modélisateur ne lui reconnaît qu'un seul attribut : son identifiant.

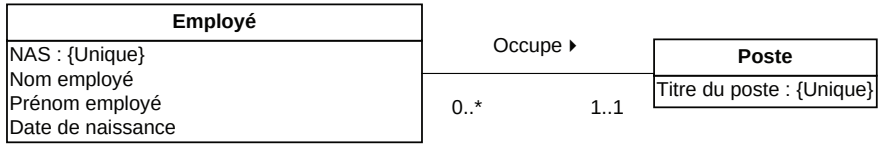
En modélisant les données pour la gestion de son personnel, une organisation pourrait juger vital de modéliser le poste occupé par l'employé. Le modélisateur devra prendre garde de confondre le tout avec une partie du tout. L'emploi du mot **Poste** pour parler du **Titre du poste** est une ambiguïté du discours que le modélisateur devra résoudre. Le **Titre du poste**,

puisqu'il représente une chaîne de caractères, par exemple « Préposé à l'entretien », est manifestement un attribut. **Poste** est d'autre part une structure plus complexe, un objet abstrait.

Le concept de **Poste** occupé par un employé représente en effet bien plus que le **Titre du poste**. Un poste peut comporter par ailleurs d'autres propriétés qui seront peut-être considérées vitales au cours du processus d'analyse des besoins. C'est le cas des attributs **Traitement annuel minimum**, **Traitement annuel maximum**, **Catégorie** (Bureau, Technicien, Professionnel, Cadre).

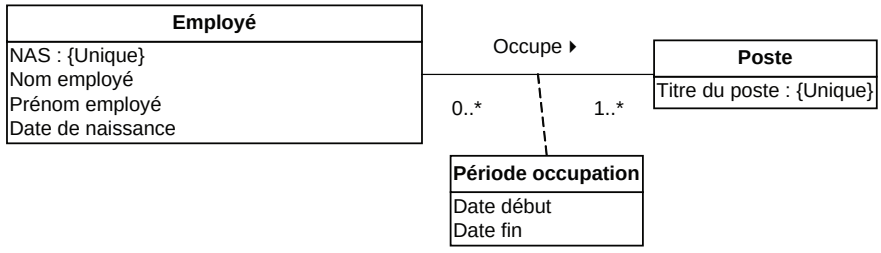
Titre du poste doit être un attribut, alors que **Poste** est forcément une entité puisqu'il se décline avec plusieurs propriétés. La règle de construction interdit de faire de **Titre du poste** un attribut de l'entité **Employé** pour illustrer le fait qu'un employé occupe un poste. **Titre du poste** est strictement un attribut de **Poste** et le modélisateur doit faire usage d'une association pour marquer le fait que l'employé occupe un poste. L'exemple 1-15 montre qu'un employé occupe un poste auquel plusieurs employés peuvent être affectés, par exemple *commis comptable*.

FIGURE 1-15 **Un employé occupe un seul poste, occupé par plusieurs employés**



Dans le contexte où le modélisateur juge vital de conserver la période durant laquelle l'employé a occupé le poste, avec les attributs **Date début** et **Date fin**, les attributs devront être liés à l'association car ils dépendent à la fois de l'employé et du poste. De plus, une multiplicité 0..***** du côté de **Poste** serait probablement plus pertinente si on souhaite représenter dans le modèle des données relatives à tous les postes occupés par un employé, pas seulement le dernier en date.

FIGURE 1-16 **Un employé occupe plusieurs postes dans sa carrière**



Principe **CHOISIR LES ENTITÉS SUR LA BASE DE CATÉGORIES PRÉDÉFINIES.** Pour faciliter le choix des entités, il est utile de consulter un répertoire des sources principales de données dans une organisation ainsi que des objets d'intérêt général pour une organisation.

Le tableau 1-3 fait état de diverses catégories d'entités pertinentes au fonctionnement d'une organisation. Ces catégories reposent sur un modèle théorique qui veut que l'environnement d'une organisation recèle des entités qui méritent d'être prises en compte, appelées entités externes, vitales au fonctionnement de celle-ci. Ce modèle montre de plus qu'elle réalise des transactions avec ces entités externes et que de telles transactions doivent être mémorisées. L'organisation dispose par ailleurs de ressources internes pour mener ses opérations appelées composants du système opérant. Ces composants doivent être inventoriés. Les biens et services résultant des opérations méritent que l'on conserve des données à leur sujet. Enfin les opérations internes conduisent à des transactions internes et à des décisions devant être conservées.

TABLEAU 1-3 **Catégories principales des entités**

Catégorie	Exemples
Entité présente dans l'environnement de l'organisation incluant des systèmes externes	Client, Fournisseur, Partenaire, Système d'autorisation pour paiement, Catalogue du fournisseur
Transaction avec une entité de l'environnement	Achat, Vente, Commande, Facture, Paiement, Réservation, Livraison, Réception, Transfert électronique de fonds, Paie
Bien ou service offert par l'organisation	Article, Prêt hypothécaire, Pièce, Vol d'avion, Menu de restaurant, Prestation de service
Composant du système opérant : personnel, rôle du personnel, équipement de production, de livraison, lieu de production, d'entreposage, ressource financière	Caissier, Employé, Machine outil, Poste employé, Pilote d'avion, Robot, Atelier, Magasin, Entrepôt, Comptoir, Caisse enregistreuse, Avion, Aéroport, Actif financier, Camion de livraison
Transaction à l'intérieur du système opérant	Contrôle qualité, Lot de production, Absence de personnel
Résultat d'une décision	Budget, Marge de crédit, Planning de travaux, Tarification, Horaire

Ces catégories sont notamment proposées dans la méthode DATARUN [PAS 90]. Elles représentent un cadre d'analyse fort utile pour repérer les entités d'un domaine ou d'un métier. Il incombe au modélisateur et à ses collaborateurs d'établir la pertinence de faire apparaître dans le modèle

conceptuel une entité appartenant à l'une ou l'autre de ces catégories, mais la démarche de la méthode DATARUN favorise aussi le repérage des entités pertinentes. Elle suggère d'analyser tout d'abord l'environnement de l'organisation et d'identifier les entités avec lesquelles l'organisation effectue des transactions. Ainsi, les entités reflétant les principales transactions externes sont définies. Les transactions externes sont généralement associées à des entités relatives aux produits et services offerts par l'organisation. Les entités décrivant les composants du système opérant qui permettent de produire les biens et services seront associées à ces derniers. Enfin les transactions internes et les décisions prises dans le cadre des opérations courantes donnent lieu aux entités pertinentes restantes.

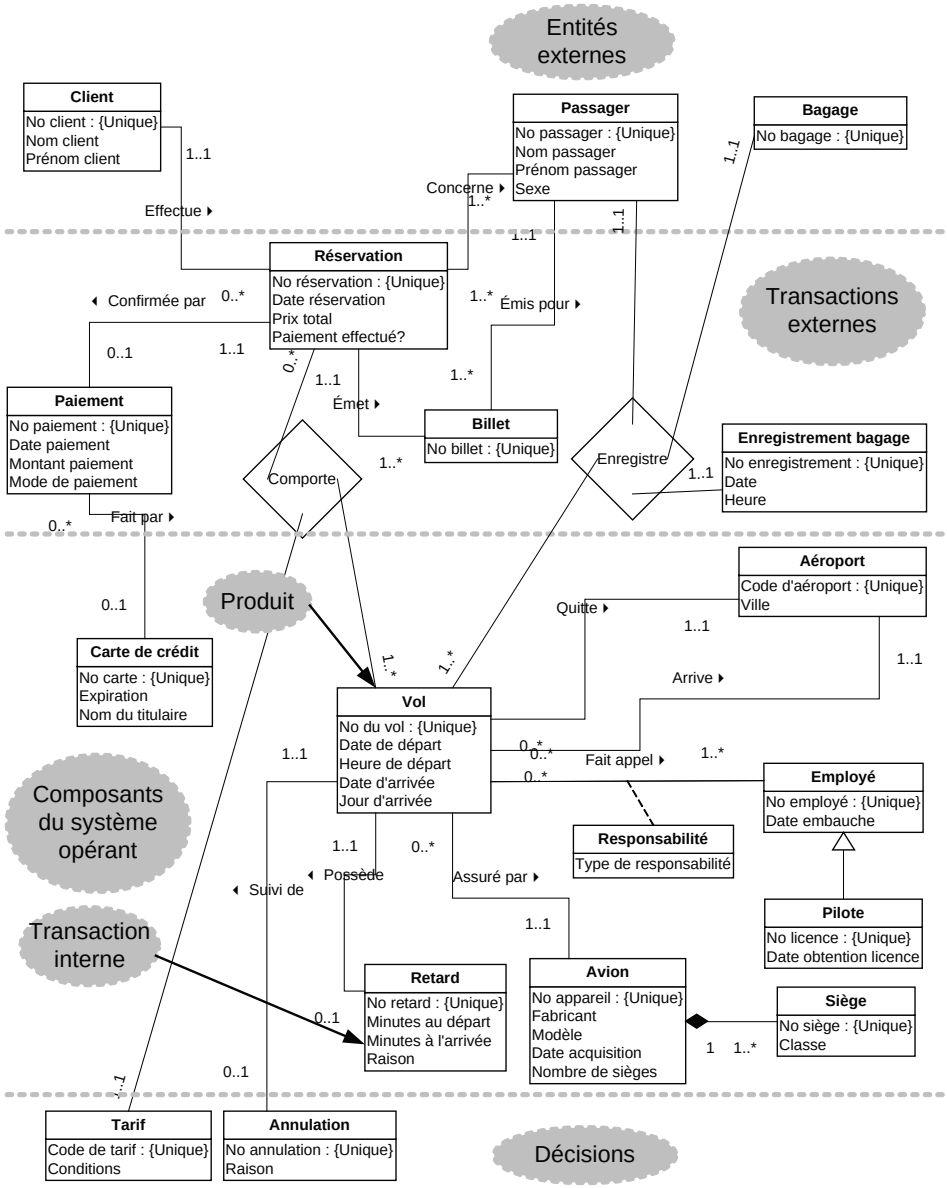
Le modèle de la figure 1-17 montre certaines entités et associations tirées de l'analyse des besoins en information d'une société de transport aérien. Le modèle est délibérément simplifié et il est forcément incomplet. Il ne vise qu'à illustrer des entités pertinentes appartenant à l'une ou l'autre des catégories du tableau 1-3. Elles reflètent par leurs attributs et leurs associations un certain nombre de données vitales au fonctionnement d'une entreprise de transport aérien.

La mise en page de ce modèle a permis de regrouper visuellement les entités par catégorie. Les entités externes dans la zone du haut, les transactions internes (**Paiement**, **Réservation**, **Billet**, **Enregistrement**) sont regroupées immédiatement en dessous. Les composants du système opérant, un produit (**Vol**) et une seule transaction interne (**Retard**) apparaissent dans la même zone. C'est un regroupement naturel car les transactions internes concernent généralement l'un ou l'autre des composants du système opérant ou les produits et services. Enfin, tout en bas on peut voir les entités représentant le résultat de décisions, ici **Tarif** et **Annulation**.

Aucune méthode, aucune règle n'impose en revanche une telle présentation. Le modélisateur a le loisir de disposer les entités et les associations selon le mode le plus approprié. La seule règle qui doit le guider est la facilité de lecture du modèle car un modèle conceptuel est aussi un outil de communication.

L'exemple 1-17 comporte des associations qui impliquent plus de deux entités, on les appelle des *associations de degré supérieur*. Nous n'avons pas insisté jusqu'ici sur ce type d'association dont le nombre est habituellement limité dans un modèle conceptuel. Les symboles pour illustrer les associations de cette nature particulière, les grands losanges, feront l'objet d'étude à la section portant sur les concepts avancés du formalisme entité-association à la fin de ce chapitre. Il en sera de même pour le petit losange noir et le triangle qui modélisent d'autres associations spécialisées.

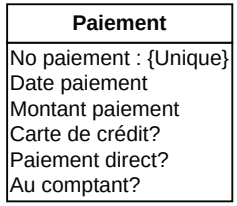
FIGURE 1-17 **Entités et associations pertinentes à un transporteur aérien**



Quelles sont les erreurs communes à éviter ?

Un risque fréquent inhérent à la modélisation conceptuelle des données est celui de confondre les valeurs prises par un attribut avec des attributs de plein droit.

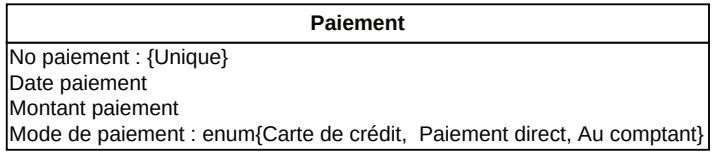
FIGURE 1-18 **Confusion « Attribut » et « Valeur d'attribut »**



Dans l'entité **Paiement** de l'exemple 1-18 par laquelle on tente de refléter les données relatives à une transaction de paiement, les attributs **Date paiement** et **Montant paiement** sont sans doute des attributs pertinents. La présence de trois attributs de type booléen (Valeur vrai ou faux), **Carte de crédit ?**, **Paiement direct ?** et **Au comptant ?**, pour spécifier selon quel mode le paiement a été effectué, est en revanche inappropriée.

Le modélisateur a choisi des attributs qui représentent chacun une des valeurs possibles d'un autre attribut qu'il n'a pas su identifier explicitement. De manière à simplifier son modèle, il aurait dû choisir un seul attribut, **Mode de paiement**, dont les valeurs sont Carte de crédit, Paiement direct et Au comptant. L'entité **Paiement** de la figure 1-19 illustre clairement le modèle voulu et la présence du type de données `enum` pour l'attribut **Mode de paiement** établit une distinction nette entre cet attribut et ses valeurs possibles.

FIGURE 1-19 **Distinction nette entre « Attribut » et « Valeur d'attribut »**



Une autre erreur commise fréquemment dans les modèles conceptuels de données consiste à y faire apparaître des associations peu pertinentes ou des associations que l'on pourrait déduire des autres associations présentes dans le modèle (association redondante).

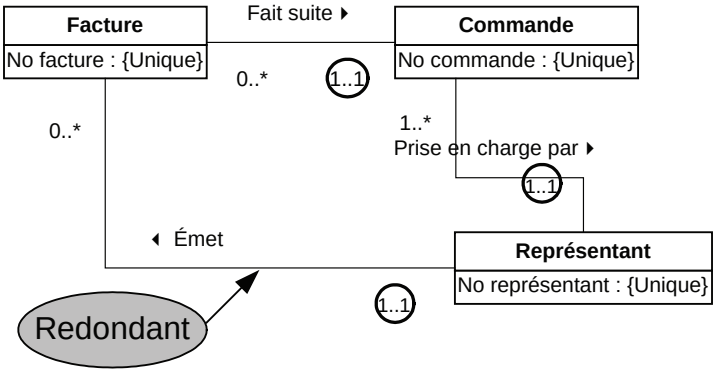
La théorie des graphes nous indique que dans un graphe comportant *n* nœuds, *(n*(n–1))/2* associations peuvent être créées entre les nœuds. Un modèle qui comporterait 6 entités pourraient théoriquement contenir jusqu'à 15 associations ! Le modélisateur doit donc les choisir avec parcimonie pour éviter que le modèle ne devienne rapidement incompréhensible car la présentation graphique du modèle devrait inévitablement comporter des croisements entre les arcs des associations. De tels croisements doivent être évités lorsque la chose est possible.

Principe **LIMITER LES ASSOCIATIONS À CELLES JUGÉES VITALES.** Une association est le reflet d'un événement qu'il est indispensable de mémoriser et qui met en cause une ou plusieurs occurrences d'entités. La pertinence de l'association est directement liée à l'importance que revêt l'événement. Ainsi l'émission d'un billet par une société aérienne crée une association avec la réservation qui lui a donné naissance et avec le passager qui pourra en bénéficier. Sur le plan conceptuel ces deux associations sont mémorables pour la société aérienne.

Dans l'exemple 1-17, il n'existe pas d'association entre l'entité **Paiement** et l'entité **Client** bien qu'il semble a priori important de savoir quel client a fait un paiement particulier. C'est que cette association est implicite. Il existe en effet une association indirecte par le biais de l'entité **Réservation** et les associations **Confirmé par** et **Effectue**. Notons que **Confirmé par** possède une multiplicité 1..1 du côté de **Réservation** et par ailleurs **Effectue** possède aussi une multiplicité 1..1 du côté de **Client**. Cela signifie qu'à partir d'un paiement, l'association **Confirmé par** mène à *une seule* réservation et cette réservation mène à *un seul* client par l'association **Effectue**. On peut donc établir qu'une association entre deux entités n'est pas vitale, car redondante, s'il existe une chaîne d'associations qui relient indirectement celles-ci et où les *terminaisons d'arrivée* de chacune des associations de la chaîne comporte une multiplicité 1..1. Une association binaire (i.e. impliquant deux entités) peut se lire dans les deux directions. La *terminaison de départ* provient de l'entité à partir de laquelle on fait une lecture et la *terminaison d'arrivée* est rattachée à l'entité sur laquelle cette lecture se termine.

C'est ainsi que dans l'exemple 1-20, l'association **Émet** qui relie une occurrence de **Facture** à une seule occurrence de **Représentant** est redondante. Il existe en effet deux associations qui relient indirectement **Facture** à **Représentant** soit **Fait suite** et **Prise en charge par**. Dans les deux cas, les terminaisons d'arrivée ont une multiplicité 1..1 (encerclées pour les mettre en évidence). Cela signifie qu'une facture mène à *une seule* commande et cette commande mène à *un seul* représentant, donc par transitivité une facture mène à *un seul* représentant. L'association **Émet** est donc injustifiée. Elle est d'autant plus injustifiée qu'elle pourrait donner lieu à une contradiction. Si

FIGURE 1-20 **Association binaire redondante**



le modélisateur avait choisi de placer une multiplicité 1..* sur l'association **Émet** côté **Représentant**, cette multiplicité serait en contradiction avec les deux autres associations qui établissent qu'une facture est associée à un et un seul représentant.

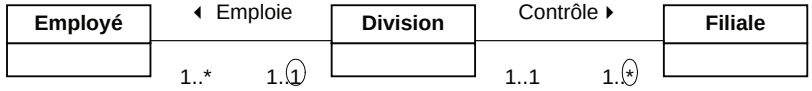
Principe **ÉVITER LES ASSOCIATIONS REDONDANTES.** Une association qui relie directement une entité **A** à une entité **B** est considérée redondante si transitivement **A** est par ailleurs associé à **B** par une chaîne d'associations dont les terminaisons d'arrivée sont toutes de multiplicité 1..1. Cette association redondante peut introduire une incohérence dans le modèle si la multiplicité sur sa terminaison d'arrivée (lecture de **A** vers **B**) est autre que 1..1.

Il faut se garder de voir une association redondante là où il n'y en a pas. Si nous revenons à l'exemple 1-17, bien qu'il existe une chaîne d'associations entre **Billet** et **Passager** par le biais de **Réservation** et des associations **Emet** et **Concerne**, celles-ci ne permettent pas d'établir un lien entre un billet et le passager inscrit sur le billet. En effet il existe une multiplicité maximale plusieurs (*) sur la terminaison d'arrivée de **Concerne**., car pour aller de **Billet** vers **Passager**, nous devons faire la lecture de cette association en partant de **Réservation** vers **Passager**. C'est pour cette raison que le modélisateur a dû formuler une association **Émis pour** permettant de lier directement un billet au passager pour lequel il a été émis. Cette association est vitale et *non redondante* car aucun attribut de l'entité **Billet** ne réfère au client et il doit en être ainsi. En vertu de la règle de construction, le nom du passager est un attribut du passager et non pas un attribut du billet. Seule une association peut rattacher le passager au billet émis.

Ces considérations sur les associations binaires dites *vitales* nous mènent à la formulation d'un nouveau principe qui recommande d'éviter le piège du genre ventilateur pour utiliser la formule de Connoly et Berg [ConB 05]. Le piège se pose lorsque trois entités **A**, **B** et **C** peuvent être associées à l'aide de deux (2) associations binaires. Si seulement deux associations sont utilisées, il existe théoriquement trois (3) façons d'associer les entités : **A-B-C**, **B-C-A** ou **B-A-C**. Dans le cas ou plusieurs de ces trois associations sont logiquement possibles et reflètent toutes trois la réalité du domaine à modéliser, il importe de choisir celle qui permet une lecture des associations avec des terminaisons d'arrivée dont la multiplicité maximale est de 1 partout.

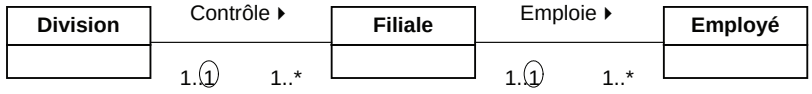
La figure 1-21 illustre une situation où le modélisateur est tombé dans le piège du ventilateur. Il tente de modéliser le lien qui existe entre **Employé**, **Division** et **Filiale**. Si on fait la lecture des associations de **Employé** vers **Filiale**, les multiplicités maximales sont de 1 côté **Division** et de ***** côté **Filiale**. Cela signifie que pour un employé on peut établir dans quelle division il travaille mais on se voit incapable d'établir dans quelle filiale car une division comporte plusieurs filiales. Il faudrait pour ce faire ajouter une troisième association entre **Employé** et **Filiale** permettant d'établir dans quelle filiale travaille l'employé. Or cette troisième association ne serait pas nécessaire si le piège du ventilateur était évité.

FIGURE 1-21 **Exemple du piège du ventilateur**



Comme le montre la figure 1-22, en plaçant **Filiale** au centre de la chaîne des associations, le modélisateur peut éviter le piège du ventilateur. Cette fois la lecture des associations de **Employé** vers **Division** montre deux terminaisons d'arrivée avec multiplicité maximale de 1 pour chacune. Cela signifie que pour un employé, il sera possible de déterminer sa filiale qui est unique et de connaître transitivement sa division sans faire appel à une troisième association.

FIGURE 1-22 **Nouveau modèle permettant d'éviter le piège du ventilateur**

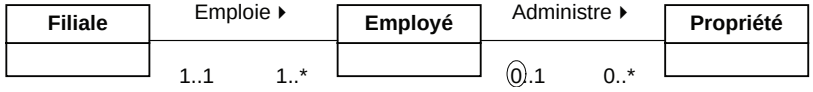


Principe **ÉVITER LE PIÈGE DU GENRE VENTILATEUR.** Dans le contexte ou deux associations binaires doivent relier trois entités, il s'agit de choisir les associations de manière à ce qu'une des deux lectures possibles de la chaîne, liant la première entité à la troisième, ait des terminaisons d'arrivée avec multiplicité maximale de 1 sur chaque association.

Un autre piège relatif aux multiplicités guette le modélisateur. Il s'agit du piège de l'abîme. Il concerne la présence de multiplicités minimales à zéro sur la chaîne de lecture des associations liant indirectement deux entités.

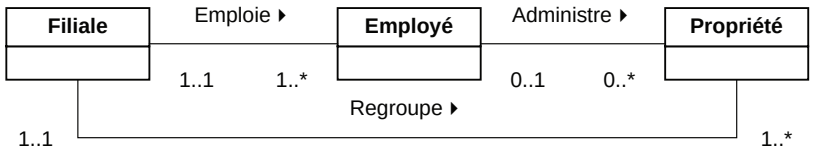
L'exemple 1-23 illustre le piège de l'abîme. Dans ce modèle il est impossible de faire le lien entre une filiale et l'ensemble des propriétés administrées. **Filiale** est associé indirectement à **Propriété** par le biais de **Employé**, or la présence d'une multiplicité 0..1 à la terminaison de départ de l'association **Administre** indique que certaines propriétés peuvent ne pas être administrées par un employé (multiplicité minimale 0). Cela signifie que, connaissant une filiale, on peut retrouver tous les employés qu'elle emploie mais comme certaines propriétés ne sont pas liées à un employé, il est impossible dans certains cas de connaître toutes les propriétés de la filiale.

FIGURE 1-23 **Modèle illustrant le piège de l'abîme**



Comme on souhaite conserver les associations **Emploie** et **Administre** qui s'avèrent toutes deux pertinentes, nous n'avons d'autre choix que d'ajouter une association directe entre **Filiale** et **Propriété**, comme le montre le modèle révisé à la figure 1-24, de manière à ce que désormais une occurrence de l'entité **Filiale** mène à toutes les occurrences de **Propriété** qu'elle regroupe.

FIGURE 1-24 **Modèle révisé évitant le piège de l'abîme**



Principe **ÉVITER LE PIÈGE DU GENRE ABÎME.** Dans le contexte ou deux associations binaires doivent relier trois entités, la présence de la multiplicité minimale à 0 sur une terminaison de départ d'une des deux associations ne permet pas de lier une occurrence de la première entité à toutes les occurrences possibles de la troisième entité. Il sera le cas échéant nécessaire d'établir une association directe entre la première et la troisième entité.

Comment nommer une entité, un attribut ou une association ?

Nous avons mentionné plus tôt qu'une entité ou un attribut devrait être nommé par un mot ou un groupe nominal. Quant aux associations binaires elles devraient être formulées par un verbe d'action et une direction de lecture. Au delà de ces conventions, un principe fondamental devrait guider le modélisateur dans le choix le plus approprié des termes d'un modèle. Nous l'appellerons à l'instar de Craig Larman [LAR 05] le *principe du cartographe*.

Principe **NOMMER LES CHOSES COMME UN CARTOGRAPHE LE FERAIT.** Tout comme le cartographe appelé à faire une carte d'un territoire doit nommer les lieux et les choses en respectant les noms en usage sur le territoire, le modélisateur doit faire appel au vocabulaire du domaine c'est à dire aux termes utilisés par les acteurs du domaine.

S'il est d'usage de parler d'un *bénéficiaire* dans le domaine de la santé, il serait inapproprié que le modélisateur utilise le mot *patient* pour faire référence à la personne qui fait l'objet de soins ou de services, même si à une époque pas très lointaine il s'agissait d'un terme largement utilisé. Il faut donc adopter le vocabulaire du domaine et de son époque.

Le cartographe porte un jugement sur la pertinence de représenter un lieu ou un objet sur une carte en fonction de l'importance que revêt le lieu ou l'objet pour les personnes qui habitent le territoire. Il en va de même pour le modélisateur. Un modèle, tout comme une carte, est une simplification du monde réel pour mieux le comprendre. Son auteur doit écarter volontairement des détails de moindre importance pour ne retenir que les éléments les plus significatifs et éviter ainsi une surcharge d'information qui pourrait au demeurant rendre le modèle incompréhensible.

CONCEPTS AVANCÉS DU FORMALISME ENTITÉ-ASSOCIATION

Comme le montre l'exemple du transporteur aérien à la figure 1-17, dans certaines circonstances le modélisateur doit faire ressortir des associations impliquant plus de deux entités, des associations de degré supérieur. De plus les associations peuvent imposer entre les entités des contraintes d'un type différent de celles vues jusqu'ici. Cette section traite de concepts qui permettent de représenter des situations ou des contraintes peu usuelles mais qu'il est néanmoins nécessaire de faire voir dans un modèle conceptuel.

Associations de degré supérieur

Les associations liant seulement deux entités, sont dites *binaires*. Celles liant plus de deux entités sont dites de *degré supérieur*. Ces dernières sont représentées à l'aide d'un grand losange où des arcs reliant les entités impliquées dans l'association sont marqués chacun de multiplicités.

Une occurrence d'une association de degré supérieur, tout comme une occurrence d'association binaire, est susceptible d'être liée à une occurrence de chacune des entités associées. Chaque arc doit disposer impérativement d'une multiplicité minimale et d'une multiplicité maximale.

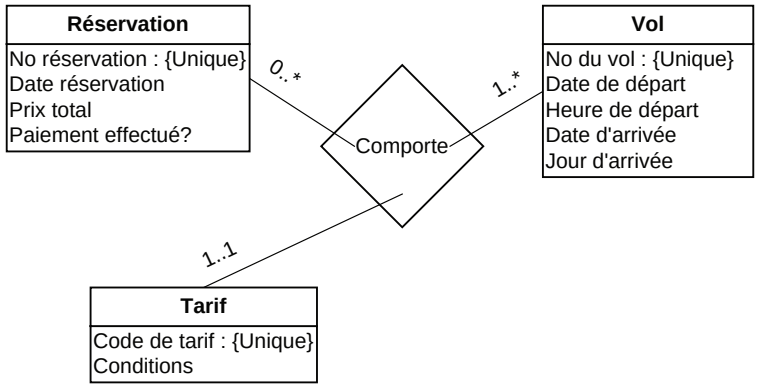
Déterminer les multiplicités dans une association de degré supérieur est un exercice intellectuel plus exigeant que celui déployé pour une association binaire.

Dans une association binaire, la multiplicité la plus près de l'entité indique combien d'occurrences de CETTE ENTITÉ peuvent être associées à UNE occurrence de L'AUTRE ENTITÉ. Dans le cas d'une association de degré supérieur ***n***, la multiplicité près d'une entité indique combien d'occurrences de CETTE ENTITÉ peuvent être associées à UNE combinaison mettant en cause une occurrence de chacune des ***n–1*** autres entités.

Essayons d'éclaircir tout cela à l'aide d'un exemple.

L'exemple 1-25 est tiré du modèle sur la société aérienne où il est stipulé qu'une réservation comporte des vols et des tarifs. Pour la compréhension du domaine, précisons qu'une réservation peut comporter plusieurs vols différents, par exemple *Québec-Montréal, Montréal-Paris, Paris-Toronto, Toronto-Québec*. Sur chaque vol de la réservation, un tarif est appliqué et celui-ci peut varier d'un vol à l'autre. Un tarif est identifié par un code qui précise les

FIGURE 1-25 **Réservation auprès d'une société aérienne**



conditions rattachées au prix du billet. Par exemple le tarif *YHJLI* pourrait avoir comme conditions : *classe économique, non remboursable, payé 30 jours avant le départ.*

Les multiplicités sont déterminées en considérant chaque entité de l'association à tour de rôle.

Multiplicité du côté **Réservation** : Combien de réservations peuvent être associées à une occurrence du couple **Tarif-Vol** ? Considérons la combinaison **Code de tarif = YHJLI** avec **No du vol = AC2033**, combien de réservations peuvent être faites avec cette combinaison ? Aucune ou plusieurs, donc multiplicité 0..*. Pourquoi ? Il est possible qu'aucune réservation n'ait été accordée avec ce code de tarif pour ce vol ou bien plusieurs peuvent l'avoir été.

Multiplicité du côté **Tarif** : Combien de tarifs peuvent être associés à une occurrence du couple **Réservation-Vol** ? Considérons la combinaison **No réservation = 4567725653** avec **No du vol = AC2033**, combien de tarifs peuvent être appliqués à cette combinaison ? Une et une seule, donc multiplicité 1..1. Pourquoi ? On ne retrouve qu'un seul tarif pour un vol dans une réservation.

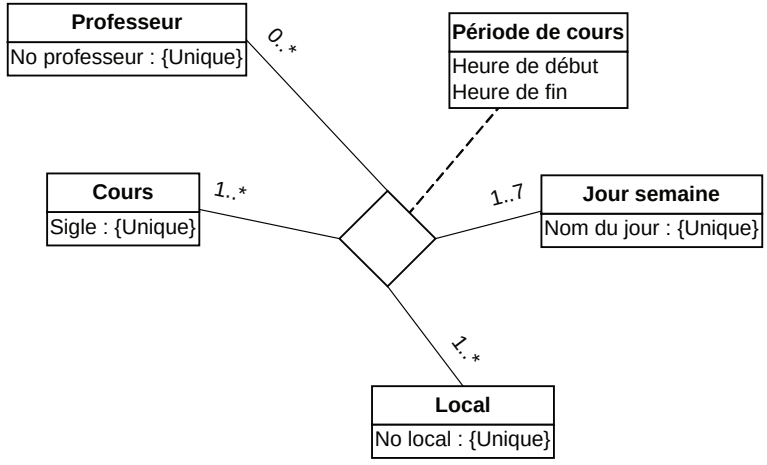
Multiplicité du côté **Vol** : Combien de vols peuvent être associés à une occurrence du couple **Réservation-Tarif** ? Considérons la combinaison **No réservation = 4567725653** avec **Code de tarif = YHJLI**, combien de vols peuvent être impliqués dans cette combinaison ? Un ou plusieurs, donc multiplicité 1..*. Pourquoi ? S'il existe un code de tarif associé à une réservation, il doit y avoir au moins un vol dans la réservation avec ce tarif ou plusieurs.

L'exercice exige donc une très bonne compréhension du domaine à modéliser. De plus le modélisateur doit pouvoir se faire une très bonne image mentale de ce que représente l'association dont une entité a été exclue ainsi que de la nature des occurrences de cette association. Une occurrence d'une association d'occurrences d'entités différentes est appelée un *couple* pour deux entités, un *triplet* pour trois et en général s'il s'agit de *n* entités on parlera d'un *tuple*. La notion de *tuple* est un concept mathématique faisant référence précisément à une association de divers objets provenant de domaines différents.

Ceci nous amène à proposer une nouvelle formulation de la logique visant à établir les multiplicités d'une association de degré supérieur ***n***: la multiplicité près d'une entité indique combien d'occurrences de CETTE ENTITÉ peuvent être associées à UN TUPLE mettant en cause une occurrence de chacune des ***n–1*** autres entités.

Une association de degré supérieur, à l'instar d'une association binaire, peut posséder des attributs. Une entité d'association est alors rattachée au losange par un trait pointillé. La figure 1-26 montre un modèle où est présente une association de degré 4, avec ses propres attributs. Elle illustre un modèle conceptuel relatif aux périodes de cours donnés par un professeur dans un établissement scolaire.

FIGURE 1-26 **Horaire des professeurs dans un établissement scolaire**



La présence de l'entité d'association **Période de cours** est justifiée car il s'agit bien d'une entité faible. En effet, **Heure de début** et **Heure de fin** d'une période de cours dépendent tous deux fonctionnellement de la combinaison

No professeur-Sigle-No local-Nom du jour. Il faut faire l'hypothèse cependant qu'un professeur ne donne qu'une seule fois un cours donné, dans un local donné, un jour donné. Par exemple si le professeur Lebrun donne le cours SIO205 le mardi dans le local A-200, le cours ne peut être donné par le professeur Lebrun que dans une seule période (avec son heure de début et son heure de fin) ce jour là dans ce local là. Autrement dit, le professeur Lebrun ne pourrait donner le cours SIO205 le mardi dans le local A-200 en deux périodes, par exemple de 8h30 à 10h30 et de 13h30 à 15h30. C'est l'hypothèse que nous devons faire pour assurer que les attributs **Heure de début** et **Heure de fin** dépendent chacun fonctionnellement de la combinaison **No professeur-Sigle-No local-Nom du jour**.

Dans le cas d'une association de degré supérieur avec attributs, l'association n'exige pas de nom. Le nom de l'entité d'association sert aussi à nommer l'association. Dans l'exemple 1-26, on indique qu'une période de cours implique un jour de la semaine, un local, un professeur et un cours. Pour reprendre l'analogie du bac, quatre bacs sont ici en cause : **Professeur**, **Cours**, **Jour semaine** et **Local** comme le montre la figure 1-27. La ficelle qui représente une occurrence de l'association possède quatre branches. Chaque branche est attachée à une fiche localisée dans un des quatre bacs. Pour que l'association existe, elle doit obligatoirement avoir quatre branches avec une fiche attachée à chaque branche provenant d'un bac différent des trois autres. Enfin la ficelle doit être étiquetée de deux valeurs : **Heure de début** et **Heure de fin**, les attributs de l'association.

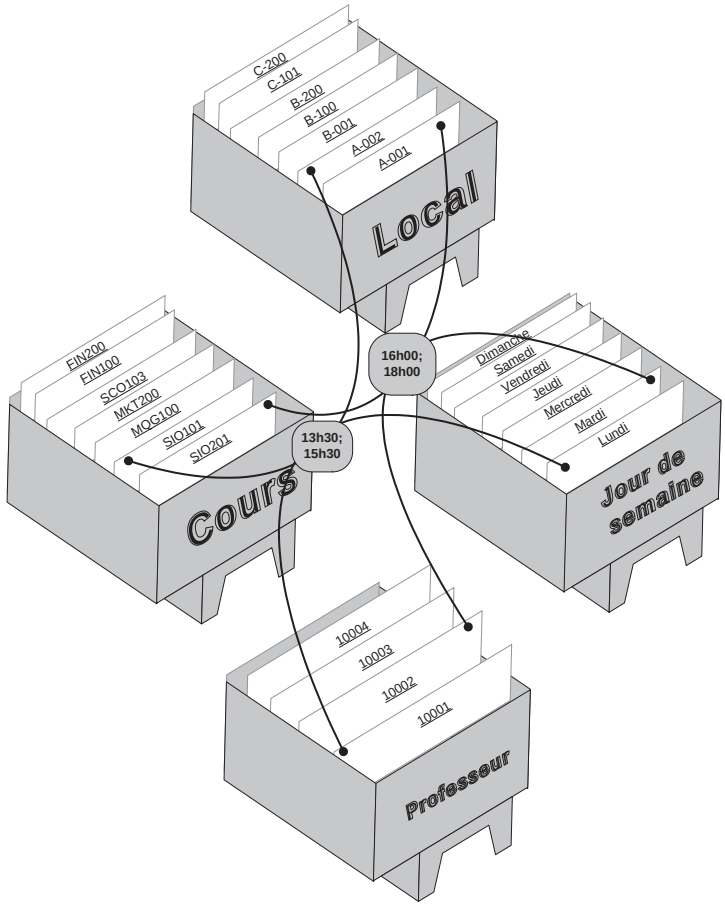
Pour simplifier la représentation visuelle du modèle, l'exemple 1-27 ne montre que deux occurrences de l'association **Période de cours**. De plus, les fiches ne comportent que la valeur de l'identifiant.

Nous complétons la présentation de cet exemple avec une justification de chacune des multiplicités de l'association **Période de cours**.

Multiplicité du côté **Professeur** : Combien de professeurs peuvent être associés à une occurrence de **Cours-Local-Jour de semaine** ? Considérons la combinaison **Cours = SIO205** avec **Local = A-033** avec **Jour de semaine = Lundi**, combien de professeurs peuvent être liés à ce tuple ? Zéro ou plusieurs, donc multiplicité 0..*. Pourquoi ? Il peut bien arriver qu'aucun professeur n'ait été identifié pour donner le cours ce jour là, ou au contraire le cours est donné ce jour là dans le même local par plusieurs professeurs, sur des périodes différentes pour éviter les conflits d'horaires.

Multiplicité du côté **Cours** : Combien de cours peuvent être associés à une occurrence de **Professeur-Local-Jour de semaine** ? Considérons la combinaison **Professeur = Lebrun** avec **Local = A-033** avec **Jour de semaine = Lundi**, combien de cours peuvent être liés à ce tuple ? 1 ou plusieurs, donc multiplicité

FIGURE 1-27 **Représentation visuelle du modèle 1-26**



1..*. Pourquoi ? Si une telle combinaison existe, elle doit exister pour au moins un cours sinon elle n'a aucun sens. Par ailleurs, elle peut concerner plusieurs cours car un professeur pourrait donner plusieurs cours le même jour dans le même local pour peu qu'ils soient tous différents et qu'il n'y ait pas de conflit entre les périodes.

Multiplicité du côté **Jour de semaine** : Combien de jours de la semaine peuvent être associés à une occurrence de **Professeur-Local-Cours** ? Considérons la combinaison **Professeur = Lebrun** avec **Local = A-033** avec **Cours = SIO201**, combien de jours de la semaine peuvent être liés à ce tuple ? 1 ou plusieurs, donc multiplicité 1..*. Pourquoi ? Si une telle combinaison existe,

elle doit exister pour au moins un jour où le cours est donné sinon elle n'a aucun sens. D'autre part le cours pourrait être donné tous les jours de la semaine, jusqu'à concurrence de 7.

Multiplicité du côté **Local** : Combien de locaux peuvent être associés à une occurrence de **Professeur-Cours-Jour de semaine** ? Considérons la combinaison **Professeur = Lebrun** avec **Cours = SIO201** avec **Jour de semaine = Mardi**, combien de locaux peuvent être liés à ce tuple ? 1 ou plusieurs, donc multiplicité 1..*****. Pourquoi ? Si une telle combinaison existe, elle doit exister pour au moins un local où le cours est donné sinon elle n'a aucun sens. Rien n'interdit qu'un professeur donne le même cours le même jour s'il est donné dans des locaux différents.

Il est souvent difficile d'établir les multiplicités minimales d'une association de degré supérieur. Selon l'interprétation qui est faite de la situation ou des hypothèses qui doivent être faites, deux choix s'offrent au modélisateur : **0** ou **1**. En cas de doute, ou de possibilités diverses d'interprétation de la situation, vaut mieux choisir celle qui représente la contrainte la moins forte soit **0**. Il en va de même des multiplicités maximales, plusieurs (*) représente une contrainte moins forte que **1**. À aucun moment, cette astuce ne doit devenir une solution de facilité en faisant appel systématiquement à des multiplicités minimales valant **0** dans des associations de degré supérieur. Dans la majorité des situations, les multiplicités peuvent être établies sans ambiguïtés si nous disposons d'une information complète de la situation. L'astuce s'applique aussi aux associations binaires.

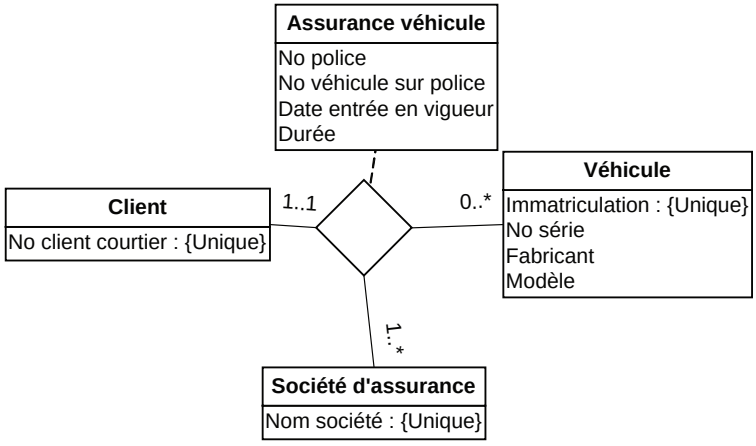
Il faut éviter de faire un usage démesuré des associations de degré supérieur. On estime qu'elles représentent moins de 20 % de toutes les associations d'un modèle et que les associations de degré 4 sont très rares. Il arrive souvent que le modélisateur non averti fasse appel à une association de degré supérieur alors qu'elle n'est pas requise. En effet dans bon nombre de situations, une association de degré supérieur peut être décomposée en quelques associations de degré moindre, notamment des associations binaires. Les conditions pour exercer la décomposition d'une association de degré supérieur font l'objet de discussion dans la section qui suit.

Décomposition des associations de degré supérieur

Il existe deux situations principales où une association de degré supérieur peut être divisée en deux associations de degré moindre. La première concerne une dépendance fonctionnelle entre les identifiants de deux entités de l'association, la seconde la présence d'une multiplicité maximale de 1 sur un des arcs de l'association. Ces deux conditions se rencontrent souvent simultanément dans une association de degré supérieur.

Considérons l'exemple 1-28. Ce modèle reflète les données vitales qu'un courtier en assurances de dommage gère pour ses clients qui s'adressent à lui pour assurer un ou plusieurs véhicules auprès d'une compagnie d'assurance. On y retrouve une association de degré 3, **Assurance véhicule**, comportant ses propres attributs. Comme toute entité d'association, **Assurance véhicule** est une entité faible. Son identifiant implicite est ici la combinaison **No client courtier-Immatriculation-Nom société**.

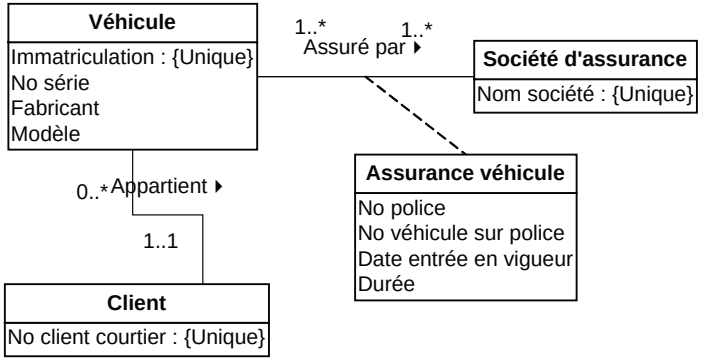
FIGURE 1-28 **Le client contracte une police d'assurance pour ses véhicules**



Par ailleurs, puisque le véhicule appartient à un seul propriétaire, ici le client, le véhicule détermine le client. Il y a donc dépendance fonctionnelle entre **Immatriculation** du véhicule et le **No client courtier**. Cette situation permet de retirer l'entité **Client** de l'association, cette dernière dépend fonctionnellement d'une autre, et de la rattacher directement à **Véhicule** par une association binaire. Seules les entités **Véhicule** et **Société d'assurance** devraient prendre part à l'association **Assurance véhicule**. Cette dernière est alors ramenée à une simple association binaire (Figure 1-29) mais pour ce faire on aura dû associer **Véhicule** et **Client**.

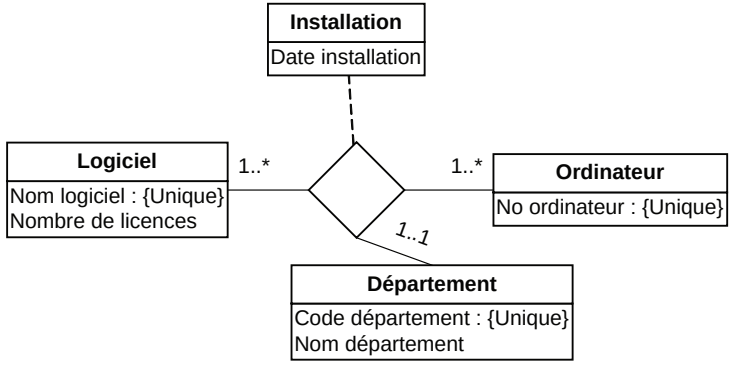
La présence d'une dépendance fonctionnelle entre deux entités liées par une association de degré supérieur peut facilement être repérée en consultant les multiplicités maximales sur les arcs de l'association. Dans l'exemple 1-28, la présence d'une multiplicité maximale de 1 côté **Client** indique que cette entité POURRAIT avoir une dépendance fonctionnelle envers une des deux autres entités, en l'occurrence **Véhicule**. Comme une multiplicité maximale valant 1 n'assure pas systématiquement une telle dépendance, il incombe au modélisateur de s'en assurer.

FIGURE 1-29 **Décomposition de l'association de degré 3 du modèle 1-28**



L'exemple 1-30 illustre un autre cas où une décomposition de l'association de degré supérieur devrait être effectuée. Il s'agit d'un modèle décrivant l'installation des logiciels sur les ordinateurs dans une organisation.

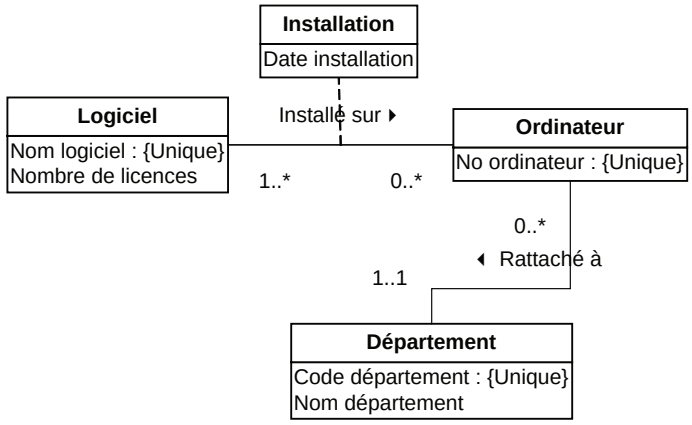
FIGURE 1-30 **Installation des logiciels sur les ordinateurs des départements**



Puisque chaque ordinateur est rattaché à UN SEUL département, le **No ordinateur** détermine le département. Il y a alors dépendance fonctionnelle entre le **Code département** et le **No ordinateur**. La présence d'une multiplicité maximale de 1 côté **Département** est un indice de ce fait. Cette situation permet d'associer directement l'entité **Ordinateur** à l'entité **Département**, ne retenant dans l'association **Installation** que les entités **Logiciel** et **Ordinateur.**

La décomposition de l'association **Installation** permet de réduire son degré et conduit à un modèle plus simple (Figure 1-31) quoique sémantiquement équivalent au modèle 1-30.

FIGURE 1-31 **Version simplifiée du modèle 1-30**



Installation est une entité faible et son identifiant implicite, la combinaison **Nom logiciel** et **No ordinateur**, respecte la règle d'identité car **Installation** représente la première installation d'un logiciel sur un ordinateur donné. On fait de plus l'hypothèse que la version fait partie du nom du logiciel, ce qui signifie que deux versions du même logiciel sont traitées comme des logiciels différents.

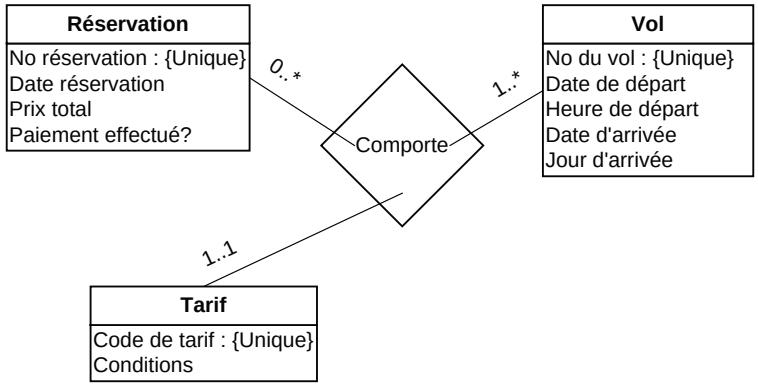
La présence d'une multiplicité maximale de 1 dans une association de degré supérieur n'est qu'un indice de dépendance fonctionnelle possible de l'entité du côté de cette multiplicité maximale de 1 envers une des autres entités associées. Il incombe au modélisateur de vérifier qu'une telle dépendance existe vraiment. Si cette dépendance peut s'avérer, le modélisateur doit décomposer l'association de degré **n** en introduisant une association binaire et une association de degré **n–1**.

Si le modélisateur n'arrive pas à établir une dépendance fonctionnelle entre deux entités d'une association comportant une multiplicité maximale de 1, cela ne signifie point que l'association ne puisse être décomposée. En effet, la présence de la multiplicité maximale de 1 dénote par définition une dépendance fonctionnelle entre une entité et la COMBINAISON DES AUTRES.

Pour illustrer cette situation, considérons à nouveau l'exemple tiré de la figure 1-17 concernant le lien existant entre une réservation, un vol et un tarif comme illustré à la figure 1-32.

Bien qu'une multiplicité maximale valant 1 se trouve côté **Tarif**, cette entité ne dépend fonctionnellement ni de **Réservation**, ni de **Vol**. En effet **Vol** ne détermine pas le **Tarif** car sur le même vol plusieurs tarifs sont applicables.

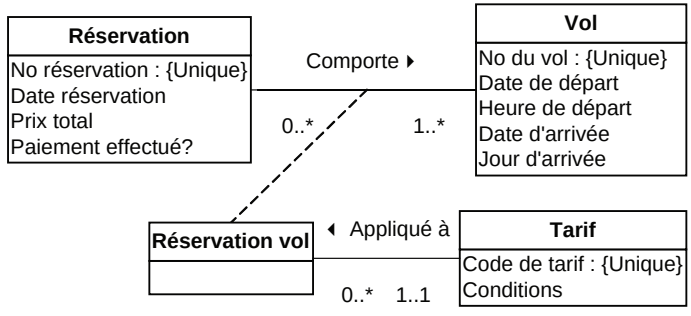
FIGURE 1-32 **Réservation auprès d'une société aérienne**



La **Réservation** ne détermine pas le tarif car une réservation comporte des vols avec des tarifs différents. Cependant **Tarif** dépend fonctionnellement à la fois de **Réservation** et **Vol**. La combinaison **Réservation-Vol** mène indubitablement à un seul tarif considérant les multiplicité 1..1 du côté de **Tarif**.

Comme le montre la figure 1-33, l'association de degré trois peut être théoriquement ramenée à deux associations binaires : une première liant les deux entités dont dépend la troisième. Cette nouvelle association aura une entité d'association artificielle sans attribut explicite qui sera associée à la troisième entité, soit **Tarif**. Par ailleurs, si une entité d'association avait été rattachée à l'association de degré 3 dans le modèle d'origine (1-32), la création d'une entité artificielle n'aurait point été requise. Il aurait suffit d'associer l'entité **Tarif** à cette entité d'association et de réduire à deux le degré de l'association initiale.

FIGURE 1-33 **Réservation ramenée à deux associations binaires en créant une entité d'association artificielle appelée *Réservation vol***



Le modèle 1-32 et la décomposition à laquelle il a donné lieu en 1-33 permettent d'étendre la stratégie de décomposition à toute association de degré **n** comme l'indique l'astuce qui suit.

La présence d'une multiplicité maximale de 1 dans une association de degré **n** représente une dépendance fonctionnelle obligatoire de l'entité du côté de la multiplicité maximale à 1 envers la combinaison des **n–1** autres entités associées. Si l'association dispose d'une entité d'association, l'entité du côté de la multiplicité maximale de 1 peut être liée directement par une association binaire à cette entité d'association. Dans le cas où une telle entité d'association n'existe guère, une entité d'association artificielle ne comportant aucun attribut peut être créée pour assurer une association binaire avec l'entité du côté de la multiplicité maximale de 1.

Quelle que soit la forme de décomposition, les multiplicités de l'association de degré **n–1** résultante doivent être réévaluées car on ne peut être assuré que les multiplicités demeurent les mêmes pour les entités restantes.

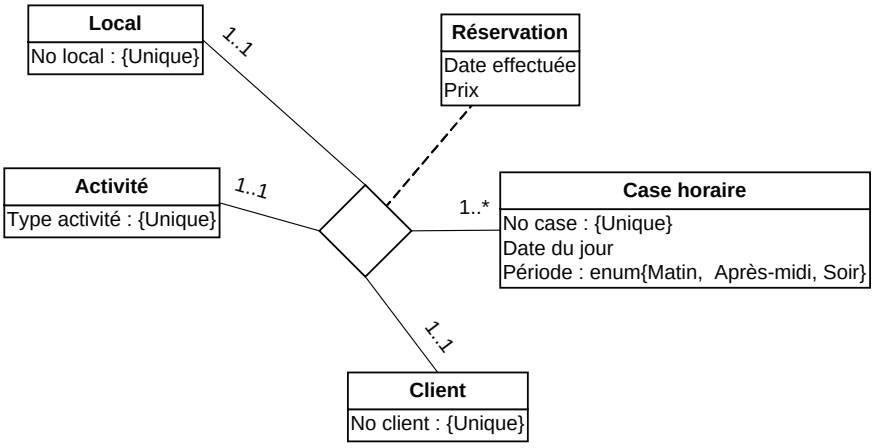
Qu'en est-il des associations de degré supérieur comportant plus d'une multiplicité maximale de 1 ? Il suffit de procéder à une première phase de décomposition, d'évaluer les multiplicités de l'association de degré supérieur résultante et de procéder à une seconde phase de décomposition si celle-ci dispose toujours d'une multiplicité maximale valant 1.

Considérons l'exemple 1-34 où on note la présence d'une association de degré 4 dotée d'une entité d'association. Comme nous le verrons, l'application des astuces données plus haut en considérant à tour de rôle, arbitrairement, chaque entité dotée d'une multiplicité maximale de 1 de son côté, conduit à une décomposition sémantiquement équivalente au modèle d'origine. Il importe cependant qu'à chacune des phases de décomposition, le modélisateur réévalue les multiplicités de l'association résultante.

L'exemple 1-34 illustre les besoins en information d'une organisation qui fait la location de ses locaux à des clients pour des types d'activités prédéterminés qui s'inscrivent dans une case horaire : une période donnée (Matin, Après-midi et Soir comme le montre le type de données énumérée) dans un jour donné. Dans ce modèle, une case horaire correspond à une période d'une journée de calendrier, par exemple le Matin du 06-06-2006.

Phase 1 : Le modélisateur peut débuter la décomposition avec une des trois entités dont la multiplicité maximale est de 1. Débutons donc arbitrairement par l'entité **Client**.

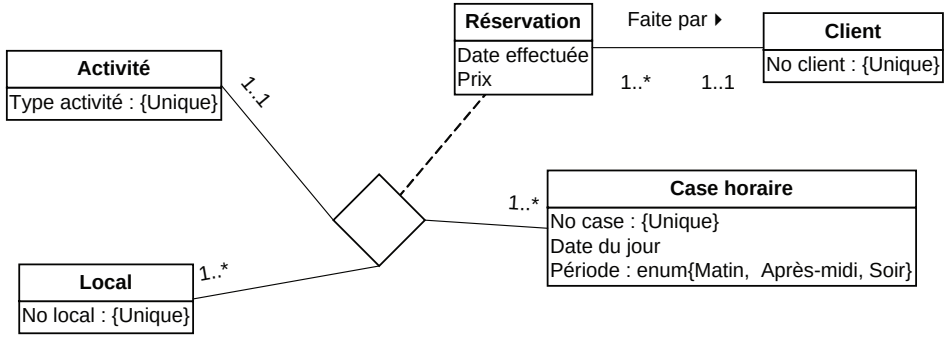
FIGURE 1-34 **Réservation d'un local par un client**



Cette entité dépend-t-elle fonctionnellement d'une des trois autres ? Aucune des trois autres, prise isolément, ne détermine le **Client**. Néanmoins, la combinaison des trois autres détermine le client, ce que montre la multiplicité 1..1 côté **Client**. L'entité **Client** peut donc être détachée de l'association pour être rattachée directement à l'entité d'association **Réservation**. La figure 1-35 montre le modèle résultant.

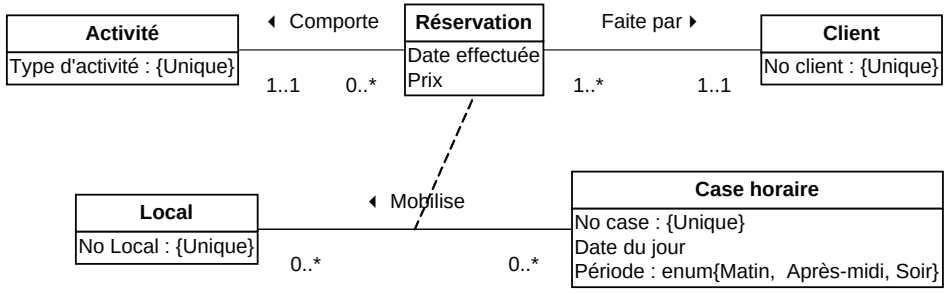
Les multiplicités de l'association de degré 3 qui résulte de cette première phase de décomposition diffèrent forcément de celle de degré 4 pour les entités restantes. Un local à un moment donné ne peut recevoir qu'une seule activité, c'est pourquoi la combinaison **Local-Case horaire** donne une multiplicité 1..1 côté **Activité**. Par ailleurs, un local et un type d'activité peuvent s'inscrire dans plusieurs périodes horaires, d'où 1..***** côté **Case horaire**.

FIGURE 1-35 **Réservation d'un local par un client, phase initiale de la décomposition**



Phase 2 : Considérons ensuite l'entité **Activité**, seule entité possédant de son côté une multiplicité maximale de 1. **Ni Activité**, ni **Case horaire** ne détermine le **Local**. L'entité **Local** ne dépend pas fonctionnellement individuellement d'une de ces deux entités mais des deux conjointement. On peut donc la détacher de l'association pour qu'elle soit rattachée tout comme **Client** à l'entité d'association **Réservation**. L'association binaire qui résulte de cette décomposition montre que les entités **Case horaire** et **Local** sont liées directement dans un contexte *plusieurs à plusieurs*. La figure 1-36 montre le résultat de la décomposition complète, en deux phases, de l'association de degré 4.

FIGURE 1-36 **Réservation d'un local par un client, phase finale de la décomposition**



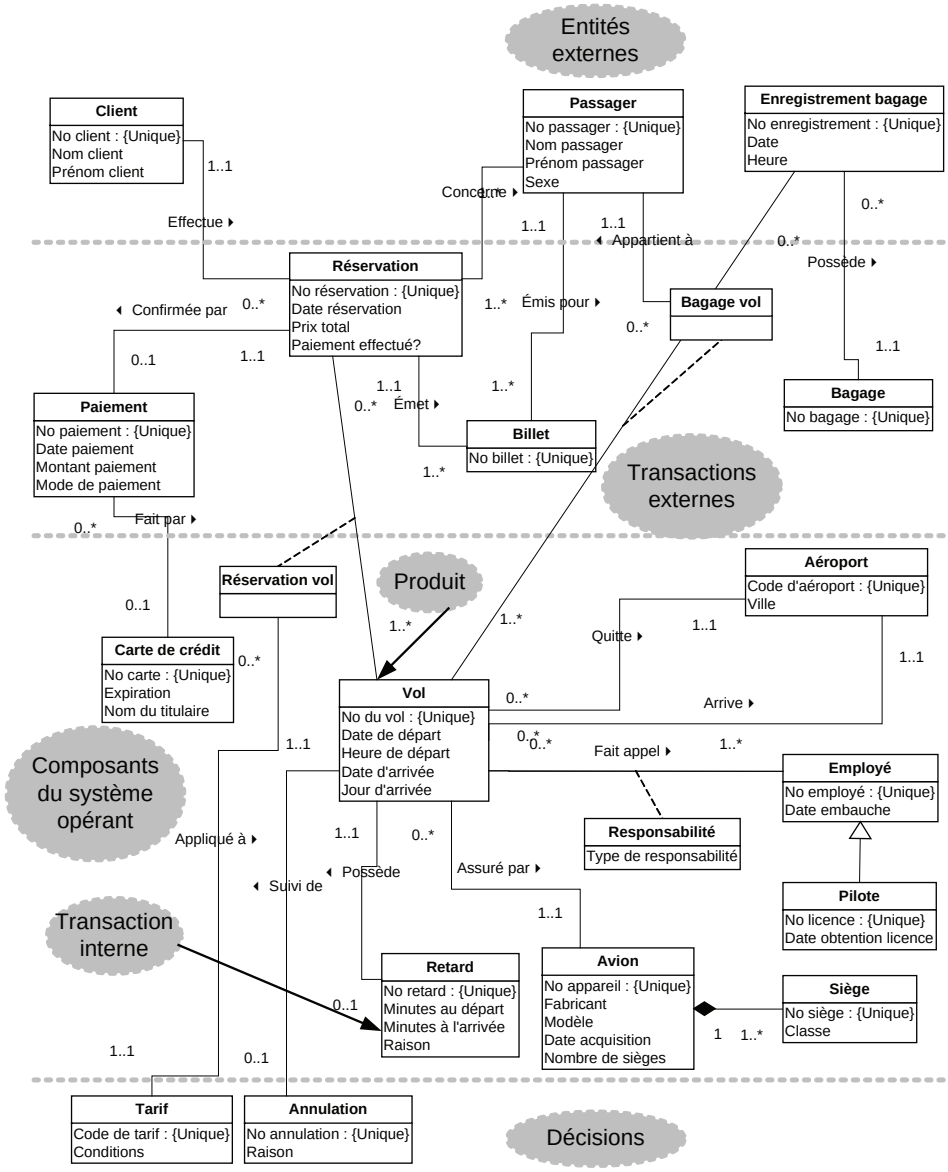
Cette section portant sur les associations de degré supérieur nous amène à conclure avec une règle encadrant l'usage de ce type d'association.

1-5 Règle de multiplicité maximale

Toute association de degré supérieur ne devrait comporter que des multiplicités maximales ***** (plusieurs). Le modélisateur peut cependant décider de conserver des multiplicités maximales de 1 dans une association de degré supérieur s'il considère que cette représentation décrit mieux le domaine, bien qu'une version où elle serait décomposée soit sémantiquement équivalente.

Nous avons décidé d'appliquer la règle au modèle 1-17 car nous considérons que les deux associations de degré supérieur, ayant chacune au moins une multiplicité maximale valant 1, n'apportent pas au modèle une meilleure expression du domaine. La figure 1-37 modélise toujours le même domaine mais cette fois SANS FAIRE APPEL à des associations de degré supérieur.

FIGURE 1-37 **Entités et associations pertinentes à un transporteur aérien**



Associations spécialisées

À l'origine, le formalisme entité-association ne comportait que des associations binaires ou de degré supérieur avec ou sans attributs. Sous l'impulsion de nouveaux formalismes introduits par la modélisation orientée objet, le formalisme EA s'est enrichi de deux nouvelles formes d'associations à vocation spécialisée : la *composition* et *l'héritage*.

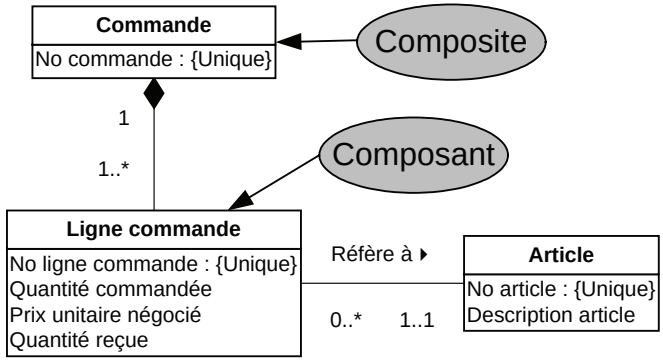
La composition

Ce type d'association est utilisé pour spécifier une dépendance *existentielle* d'une entité envers une autre. Autrement dit, elle permet d'établir qu'une entité n'existe qu'en fonction de l'existence d'une autre, comme une partie est un élément d'un tout indissociable. Une partie ne peut exister sans un tout. Pour utiliser un exemple un peu macabre, les doigts d'une main ne peuvent exister isolément, ils n'existent que dans la main à laquelle ils appartiennent.

L'association de composition est symbolisée par un petit losange noir et spécifie une relation entre un composant et un composite, une partie et un tout. On peut systématiquement lire l'association en utilisant les groupes verbaux **Est composé de** ou **Fait partie de** selon la direction empruntée pour la lecture. C'est la raison pour laquelle une association de composition ne porte jamais de nom explicite.

L'exemple 1-38 montre une association de composition qui impose une contrainte de dépendance existentielle entre une commande et les lignes de la commande. Une ligne de commande schématise les données relatives à

FIGURE 1-38 **Composition et dépendance existentielle**



chaque article commandé. Une occurrence d'une ligne de commande ne peut exister sans une association à une commande. La disparition d'une commande justifie la disparition de toutes les lignes de la commande. C'est ce qu'on entend par une dépendance existentielle entre ligne de commande et commande.

Il peut être utile de numéroter chaque ligne de la commande pour y faire référence individuellement, notamment pour savoir combien d'articles ont été reçus. C'est ce que démontre l'exemple 1-38.

Association de composition ▸ Type d'association représentant une contrainte de coïncidence de vie très forte entre le tout et sa partie. L'entité qui représente le tout est appelée **composite**, l'entité qui représente une partie du tout est appelée **composant**. La création d'une occurrence d'un composant exige qu'un composite existe déjà pour s'y associer. La fin de vie d'une occurrence de composite entraîne en cascade la fin de vie de toutes les occurrences des composants associés. (*Composition*)

Dans une association de composition, la multiplicité du côté du composite est systématiquement 1..1, ou simplement 1, car un composant doit appartenir à un et un seul composite. L'identifiant du composant est choisi dans le contexte du composite. Le choix de l'identifiant d'un composant doit garantir qu'il sera possible de distinguer les occurrences du composant associées à la même occurrence du composite. Dans l'exemple 1-38 chaque **No ligne commande** est distinct dans une même commande ; il s'agit d'un bon identifiant pour un composant. Cependant, si **Commande** et **Ligne commande** étaient associées par une association binaire simple, **No ligne commande** ne serait pas un bon choix : la ligne numéro 1 est présente dans plusieurs commandes. L'association de composition est la seule où la règle d'identité peut être transgressée au niveau de l'entité composant : deux occurrences d'une entité peuvent avoir la même valeur pour leur identifiant respectif si et seulement si elles appartiennent à deux composites différents. Théoriquement, l'identifiant réel d'un composant est la combinaison de l'identifiant du composite et l'identifiant apparaissant dans ce composant.

Trois questions peuvent être soulevées pour déterminer si une entité **B** est un composant de l'entité composite **A** : L'entité **A** est-elle composée de l'entité **B** ? Si oui, une occurrence de l'entité **B** appartient-elle de manière exclusive à une occurrence de l'entité **A** ? La durée de vie d'une occurrence de l'entité **A** détermine-t-elle la durée de vie de toutes les occurrences associées de l'entité **B** ?

Ainsi, dans l'exemple 1-38, Une commande est forcément constituée d'une ou plusieurs lignes commandes. D'autre part une ligne commande doit obligatoirement être associée à une et une seule commande. Enfin, la destruction d'une commande entraîne automatiquement la destruction de toutes les lignes de la commande.

L'héritage

L'association *d'héritage* est employée d'abord et avant tout pour assurer le respect d'une règle de modélisation appelée la *règle d'homogénéité*. Nous avons pris le parti d'introduire cette nouvelle règle à ce moment-ci pour mettre en relief l'usage particulier d'une association d'héritage.

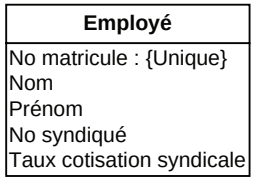
1-6 Règle d'homogénéité

Tous les attributs d'une entité sont pertinents à cette entité et éventuellement tous doivent posséder une valeur. La seule raison qui fasse qu'une occurrence n'ait pas de valeur pour un attribut, c'est que cette valeur ne sera connue que plus tard.

La règle d'homogénéité ne permet pas qu'il y ait un groupe d'occurrences de l'entité pour lesquelles un attribut ne puisse avoir de valeur. Elle interdit d'autre part qu'une entité ait des attributs *mutuellement exclusifs*, c'est-à-dire qu'une occurrence possédant une valeur pour un attribut ne puisse dès lors avoir de valeur pour un second attribut. La présence de valeur chez l'un excluant la présence d'une valeur chez l'autre.

L'exemple 1-39 montre une entité qui ne respecte manifestement pas cette règle. Puisque certains employés ne sont pas syndiqués, notamment les employés cadres, les occurrences de l'entité représentant des employés non syndiqués ne pourront avoir de valeur pour **No syndiqué** et **Taux cotisation syndicale**.

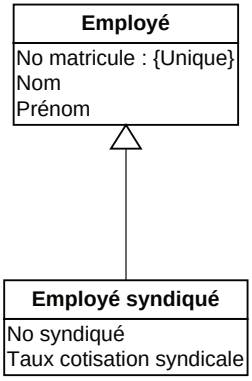
FIGURE 1-39 **Transgression de la règle d'homogénéité**



Il n'y a pas homogénéité du concept **Employé**. Certaines occurrences ont des valeurs pour des attributs pour lesquels d'autres occurrences ne peuvent avoir de valeur. Pour éviter de transgresser la règle d'homogénéité, il faut généralement montrer dans le modèle conceptuel qu'une entité peut être un cas particulier d'une autre, la forme spécialisée de cette dernière. L'association d'héritage spécifie qu'une entité *est une sorte* d'autre entité, donc un cas particulier de cette dernière. Par exemple l'entité **Employé syndiqué** peut se définir comme *une sorte de* **Employé**. L'héritage est symbolisé par un petit triangle isocèle dont le sommet pointe sur l'entité qui représente le cas général, ou le *supertype*, et qui, à sa base, est relié par un arc à l'entité qui représente le cas particulier, le *sous-type*.

La figure 1-40 montre un modèle comportant deux entités liées par une association d'héritage. Dans cet exemple, on peut lire qu'une entité **Employé syndiqué** est une sorte d'entité **Employé** et qu'elle hérite de tous les attributs de **Employé**. Désormais, toute occurrence de **Employé** possède une valeur pour chacun de ses 3 attributs et dans le cas de **Employé syndiqué** chaque occurrence possède une valeur pour les 5 attributs, car une entité qui hérite d'une autre entité hérite de tous ses attributs. Les attributs d'un sous-type sont la combinaison des attributs de ses supertypes.

FIGURE 1-40 **Héritage et respect de la règle d'homogénéité**

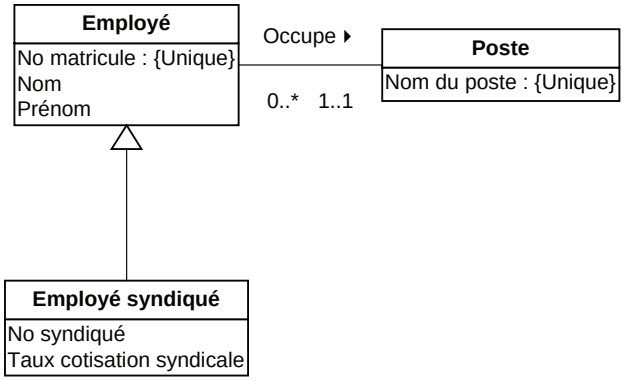


L'association d'héritage ne comporte pas de multiplicité car elle ne constitue par une contrainte sur le nombre d'occurrences qui y prennent part, mais bien sur la structure des entités. Son nom tout comme pour la composition est implicite : dans ce cas ce pourrait être **Est une sorte de**. L'entité qui hérite des attributs d'une autre n'a pas d'identifiant qui lui est propre car elle hérite aussi de l'identifiant de son supertype.

Association d'héritage Type d'association qui définit la structure d'une entité en fonction d'une autre. Une entité appelée le **supertype** identifie les attributs communs, une autre précise les attributs spécifiques à un **sous-type** de la première ; le sous-type hérite à la fois des attributs, dont l'identifiant, et des associations de son supertype (*Inheritance*).

L'exemple 1-41 montre une association entre l'entité **Employé** et l'entité **Poste**. Non seulement l'entité **Employé syndiqué** hérite des attributs de **Employé**, elle hérite aussi de l'association avec **Poste**. On pourra dire qu'un employé syndiqué est une sorte d'employé et que tout employé occupe un poste. En ce sens **Employé syndiqué** est un sous-type de l'entité **Employé**.

FIGURE 1-41 **L'héritage porte sur les attributs et les associations**

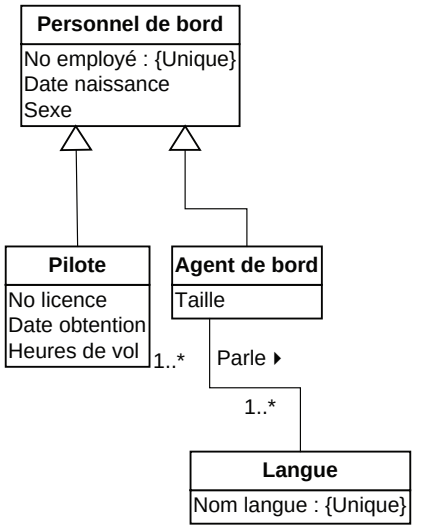


Plusieurs entités peuvent hériter de la même entité. Le modèle 1-42 spécifie que le personnel de bord se divise en deux catégories : les pilotes et les agents de bord avec des attributs qui sont propres à chaque catégorie. Une association propre aux agents de bord permet de savoir quelles sont les langues parlées par l'agent. On considère essentiel de savoir quelles langues peut parler un agent, mais ceci n'est pas jugé vital pour un pilote.

Une association d'héritage est aussi appelée association de *généralisation/spécialisation* dans la notation UML.

Tout comme les associations de degré supérieur, les associations de composition et d'héritage sont relativement rares. La majorité des modèles conceptuels n'en comporte aucune. Ce sont des associations à usage très spécialisé pour exprimer des situations exceptionnelles.

FIGURE 1-42 **Deux entités héritent de la même entité**



Nous complétons cette section en présentant et illustrant un certain nombre de contraintes entre les associations, contraintes qui permettent d'enrichir la sémantique d'un modèle conceptuel mais qui n'ont pas à apparaître obligatoirement dans ce type de modèle. En effet, un modèle conceptuel peut être considéré *complet* sans que ne soient spécifiées des contraintes entre les associations. Un modèle conceptuel est complet s'il comporte :

1. une ou plusieurs entités,
2. un identifiant pour chaque entité, excepté pour une entité faible et un sous-type où l'identifiant est implicite
3. des associations entre certaines entités
4. des multiplicités sur chaque terminaison de toutes les associations, sauf celles de composition où les multiplicités sont implicites et celles d'héritage où les multiplicités ne sont pas pertinentes.

Contraintes entre les associations

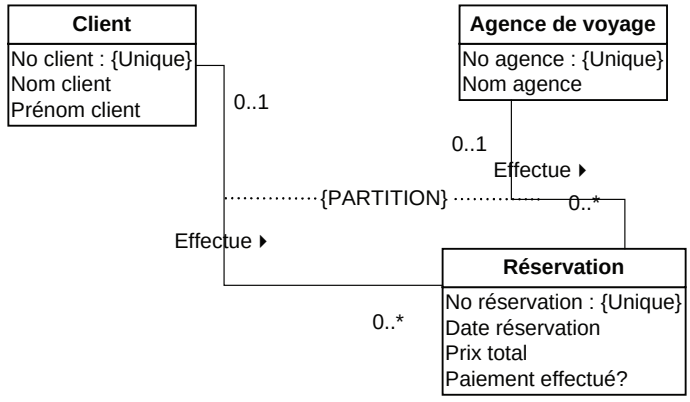
Les contraintes formulées dans un modèle conceptuel ne concernent pas seulement un attribut (contrainte sur le domaine) ou une association (contrainte de multiplicité). Dans certains cas exceptionnels une contrainte peut impliquer des associations. On parle ici de *contraintes inter-associations* qui s'appliquent entre des occurrences de plusieurs associations partageant des entités.

Contrainte de partition

Une contrainte de partition est employée pour spécifier que les occurrences d'une entité partagée ne peuvent participer simultanément à deux associations : la participation à une association exclut la participation à une autre, bien que la participation soit OBLIGATOIRE à l'une OU à l'autre.

La figure 1-43 illustre un tel type de contrainte. Les entités et associations en cause sont reprises du modèle 1-37 portant sur les données vitales d'une société aérienne. Le modélisateur a voulu préciser qu'une réservation peut être faite par un client directement ou par une agence de voyage, PAS LES DEUX À LA FOIS, car il s'agit des deux seuls moyens de faire une réservation mais le client ne doit faire appel qu'à un des deux moyens pour effectuer sa réservation. La contrainte inter-association est symbolisée par un trait pointillé qui fait le lien entre les arcs des deux associations. Le trait est étiqueté par un mot entre accolades indiquant la nature de la contrainte, ici la PARTITION.

FIGURE 1-43 **Contrainte de partition**

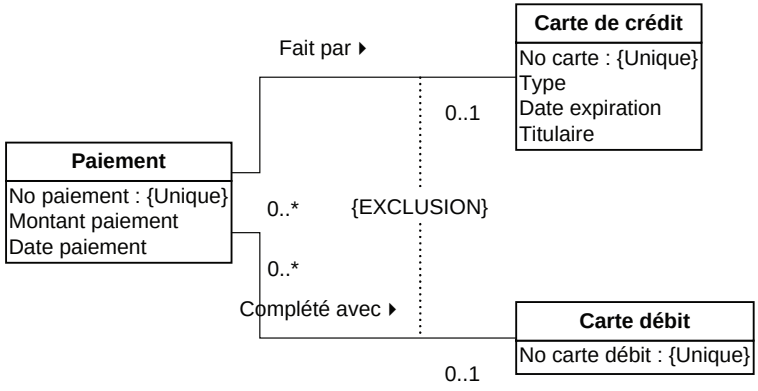


Contrainte d'exclusion

Une contrainte d'exclusion est un cas particulier de la contrainte de partition. Tout comme cette dernière, une contrainte d'exclusion stipule que la participation aux associations d'une occurrence de l'entité partagée est mutuellement exclusive. La différence est que la participation à l'une ou l'autre des associations N'EST PAS OBLIGATOIRE.

La figure 1-44 illustre une contrainte d'exclusion. Un paiement est fait soit par carte de crédit, soit par carte de débit, pas les deux à la fois. Cependant le paiement peut être fait par un autre moyen, par exemple au comptant. La participation d'une occurrence de **Paiement** à l'une ou l'autre des associations N'EST DONC PAS OBLIGATOIRE.

FIGURE 1-44 **Contrainte d'exclusion**

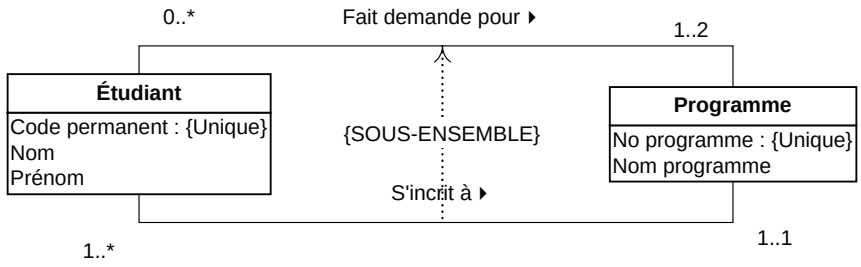


Contrainte d'inclusion

Une contrainte d'inclusion est employée pour spécifier que les occurrences d'une association sont aussi les occurrences d'une autre association liant les mêmes entités. Le modèle 1-45 comporte deux associations binaires sur les mêmes entités.

Le modèle spécifie qu'un étudiant peut faire une demande d'admission dans une université où il indique un maximum de deux programmes. L'inscription d'un étudiant ne peut concerner que l'un des programmes pour

FIGURE 1-45 **Contrainte d'inclusion**

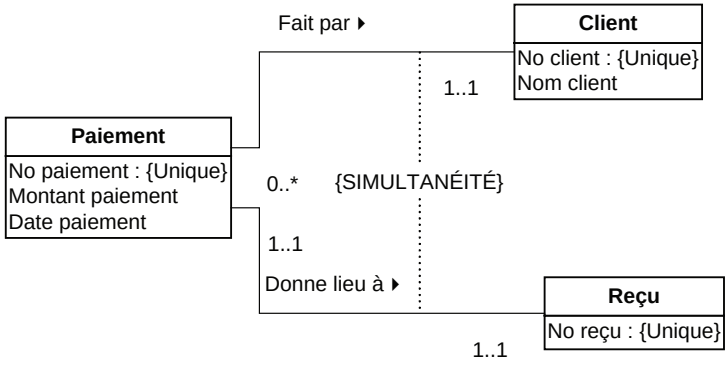


lesquels une demande d'admission a été faite par l'étudiant. Une contrainte d'inclusion est symbolisée par une flèche en pointillé qui pointe vers l'association dont les occurrences de la première constituent un sous-ensemble de l'autre, avec une étiquette portant la mention {SOUS-ENSEMBLE}.

Contrainte de simultanéité

Une contrainte de simultanéité est placée entre deux associations liant les mêmes entités pour préciser que l'existence d'une occurrence d'une association entraîne nécessairement la présence d'une occurrence de l'autre association. Comme l'indique le modèle 1-46, le paiement effectué par un client entraîne nécessairement l'émission d'un reçu relatif au paiement.

FIGURE 1-46 **Contrainte de simultanéité**

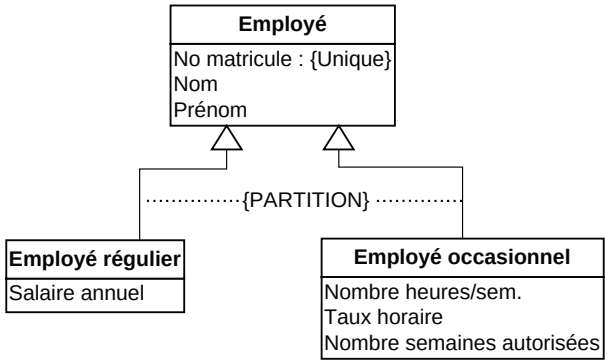


Contrainte de partition sur une association d'héritage

Une contrainte de partition peut aussi être spécifiée sur une association d'héritage. Dans un tel cas elle permet de préciser QU'AUCUNE occurrence du supertype ne peut exister. Seules des occurrences des sous-types sont acceptables et une occurrence NE PEUT appartenir aux deux sous-types à la fois.

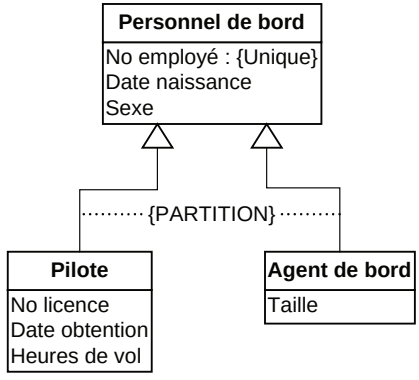
Supposons que dans une organisation, un employé puisse avoir soit un statut de *régulier*, soit *d'occasionnel* mais pas les deux à la fois. Si le modélisateur définit une entité **Employé** pour les attributs communs à tous les employés et deux entités qui héritent des attributs de celle-ci pour représenter les statuts, une contrainte de partition permet d'établir que TOUS les employés ont l'un ou l'autre des statuts, mais PAS LES DEUX À LA FOIS. La figure 1-47 illustre cette condition.

FIGURE 1-47 **Contrainte de partition sur l'héritage**



L'exemple sur le personnel de bord donné plus tôt peut être raffiné à l'aide d'une contrainte de partition sur une association d'héritage. **Pilote** et **Agent de bord** sont clairement des catégories mutuellement exclusives. Aussi proposons-nous à la figure 1-48 un modèle plus conforme à la réalité car en l'absence de contraintes sur les associations d'héritage, le supertype peut avoir des occurrences qui n'appartiennent ni à l'un ni à l'autre des sous-types et il peut exister des occurrences qui appartiennent aux deux sous-types à la fois. Dans le cas qui nous occupe, un pilote ne peut être un agent de bord et vice versa. Chaque personnel de bord appartient à une des deux catégories exclusivement. D'où l'importance de placer une contrainte de partition entre les liens d'héritage si on souhaite mettre en évidence cette particularité.

FIGURE 1-48 **Partition entre les types Pilote et Agent de bord**

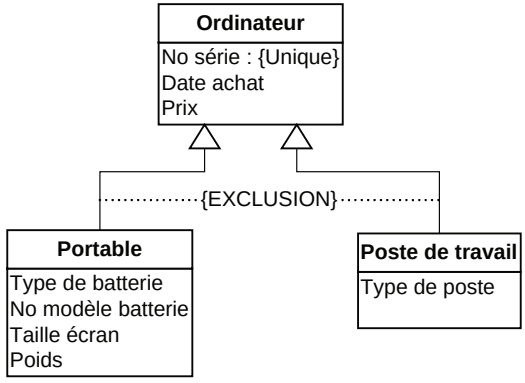


Contrainte d'exclusion sur une association d'héritage

Une contrainte d'exclusion, au contraire d'une contrainte de partition, n'interdit pas que l'entité supertype puisse avoir des occurrences. En revanche, tout comme la contrainte de partition, elle indique que les sous-types sont mutuellement exclusifs.

Le modèle 1-49 spécifie les attributs d'un ordinateur portable et d'un poste de travail. Un ordinateur peut être d'un de ces deux types, pas des DEUX À LA FOIS, mais peut aussi être D'UN AUTRE TYPE, non défini dans le modèle. Par exemple un mini-ordinateur. Tous les types d'ordinateurs non définis dans le modèle auront comme attributs ceux du supertype **Ordinateur**.

FIGURE 1-49 **Contrainte d'exclusion sur l'héritage**

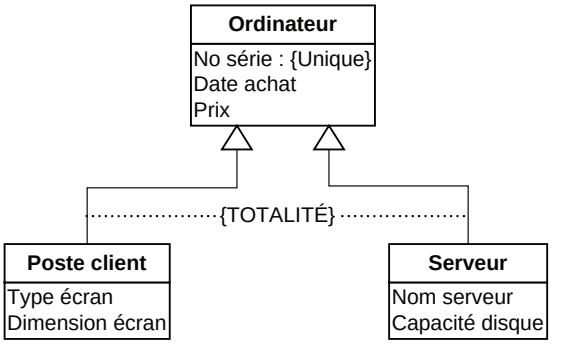


Contrainte de totalité sur une association d'héritage

Si l'on souhaite interdire les occurrences du supertype tout en spécifiant qu'une occurrence peut être des deux sous-types à la fois, la contrainte de totalité sera utilisée. Elle indique que les sous-types représentent la totalité des occurrences acceptables. En conséquence de quoi aucune catégorie d'ordinateur – autres que **Poste client** ou **Serveur** – n'est acceptable dans ce modèle.

Dans l'exemple 1-50, il est stipulé qu'un ordinateur est utilisé soit comme **Poste client** soit comme **Serveur** et, dans certains cas, pour les deux fonctions à la fois. Aucune autre catégorie d'ordinateur n'existe.

FIGURE 1-50 **Contrainte de totalité sur l'héritage**



CAS AVANCÉS DE MODÉLISATION CONCEPTUELLE DES DONNÉES

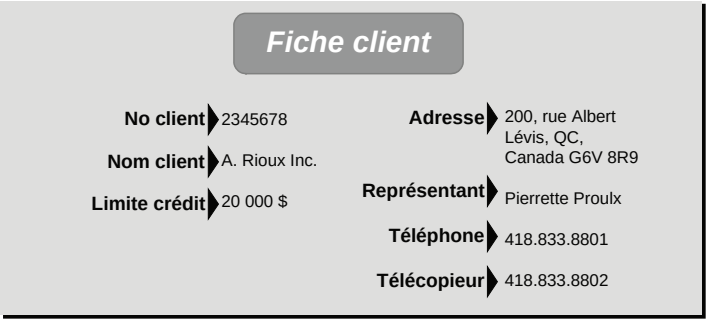
Les études de cas présentées dans cette section reflètent à bien des égards la démarche suivie par un modélisateur professionnel lorsqu'il s'attaque à un problème de modélisation. Comme nous le verrons de manière détaillée au chapitre 4, le modélisateur repère une bonne part des données d'intérêt pour un modèle conceptuel à travers des documents en format papier utilisés dans le cadre du fonctionnement de l'organisation. C'est pourquoi les cas qui suivent comportent tous un exposé du cas qui fait référence à un ou des documents. Les données à modéliser pour chaque cas sont présentes dans ces documents.

Recherche des structures de données dans un document

Le travail du modélisateur débute par un inventaire des documents pertinents au domaine à modéliser. Il dresse ensuite une image schématique du contenu de chaque document appelée le *descriptif du document*. Le descriptif du document permet de voir d'un coup d'œil les données présentes dans le document ainsi que les *structures de données*. Une structure de données est un groupe de données que l'on retrouve généralement ensemble sur un document et auquel on peut faire référence par un nom ou un groupe nominal. Par convention, une structure de données est inscrite sur le descriptif du document en caractères gras.

Le document suivant, appelé **Fiche client**, permet à une entreprise de conserver des données sur chacun de ses clients.

FIGURE 1-51 **Document « Fiche client »**



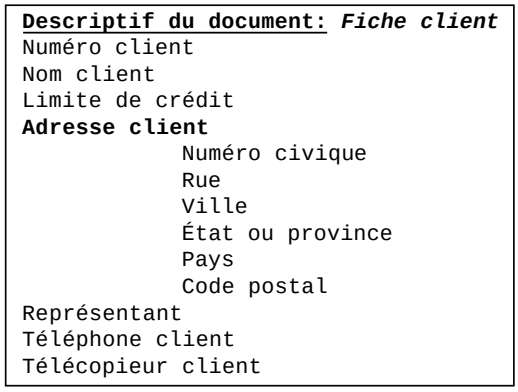
Le modélisateur avisé doit pouvoir distinguer les données élémentaires et les groupes de données. Pour chaque groupe de données bien identifié, il devra créer une structure de données. La structure devra porter un nom et elle comportera une liste de données élémentaires.

L'adresse sur la fiche client représente un groupe de données élémentaires : numéro civique, rue, ville, pays et code postal. Le numéro de client est aussi une donnée élémentaire, comme toutes les autres données du document à l'exception de l'adresse. Le modélisateur doit donner un nom aux données élémentaires qui soit le moins ambigu possible car la donnée pourrait être considérée vitale. Elle apparaîtra alors dans un modèle conceptuel où une autre donnée, différente, pourrait porter le même nom. Cette situation doit être évitée. Toute donnée vitale doit porter un nom qui puisse la distinguer des autres. Les synonymes sont donc à proscrire.

Considérons le cas de la donnée **Téléphone**. Dans la fiche client, cette donnée réfère clairement au numéro de téléphone du client. Dans une fiche sur les fournisseurs, **Téléphone** devrait référer au numéro de téléphone du fournisseur. Pour éviter toute ambiguïté le modélisateur devrait nommer **Téléphone client** la donnée présente sur la fiche client.

Le descriptif du document **Fiche client** créé par le modélisateur est donné à la figure 1-52.

FIGURE 1-52 **Descriptif du document « Fiche client »**



Comme nous le mentionnions plus haut une structure de données est inscrite en gras dans le descriptif. De plus, les données constituant la structure sont placées immédiatement au-dessous du nom de la structure avec un décalage vers la droite mettant bien en évidence le lien d'appartenance à la structure de données.

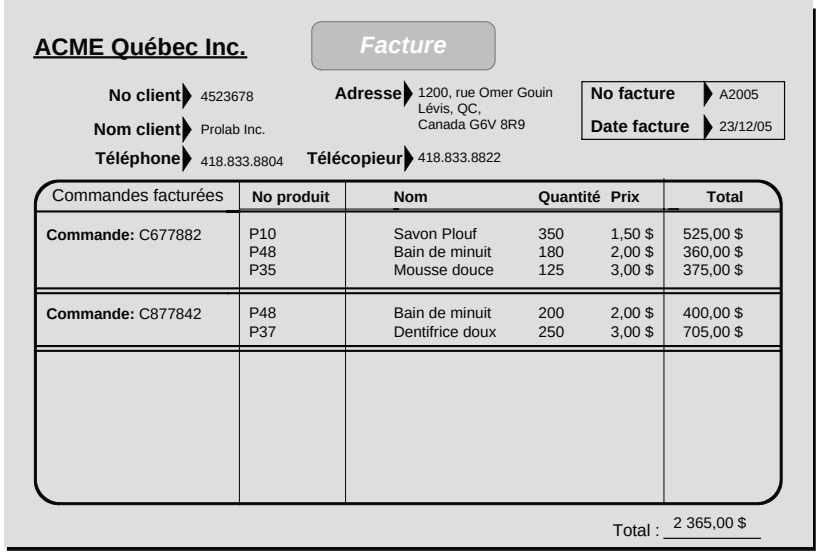
D'autres conventions doivent être respectées. Une structure de données peut contenir une autre structure de données. On parle alors de structures *emboîtées*. Dans ce cas, le nom de la structure doit aussi être décalé vers la droite par rapport à la structure à laquelle il appartient, tout comme les autres données élémentaires. De plus, une structure peut être répétée dans un document. Un astérisque est alors inscrit entre parenthèses à la suite du nom de la structure.

Le document suivant comporte à la fois des structures emboîtées et des structures répétées. Il s'agit d'une facture qui fait référence à plusieurs commandes (structure répétée), chaque commande possède (emboîte) plusieurs lignes (structure répétée), soit une ligne par produit commandé.

Le document de la figure 1-53 reprend des données déjà répertoriées dans le descriptif du document **Fiche client**. Puisque les deux documents sont rattachés au même domaine de modélisation, il est bien sûr hors de question de nommer différemment les données et structures de données déjà répertoriées : **Numéro client**, **Nom client**, **Téléphone client**, **Télécopieur client** et **Adresse client**.

Par ailleurs, on voit clairement que la facture fait référence à des commandes. La donnée qui caractérise une commande est d'abord le numéro de commande et ensuite les produits d'une commande. Le modélisateur a

FIGURE 1-53 **Document « Facture »**



ACME Québec Inc.

Facture

No client 4523678
Nom client Prolab Inc.
Téléphone 418.833.8804

Adresse 1200, rue Omer Gouin
Lévis, QC,
Canada G6V 8R9

Télécopieur 418.833.8822

No facture A2005
Date facture 23/12/05

Commandes facturées	No produit	Nom	Quantité	Prix	Total
Commande: C677882	P10	Savon Plouf	350	1,50 $	525,00 $
	P48	Bain de minuit	180	2,00 $	360,00 $
	P35	Mousse douce	125	3,00 $	375,00 $
Commande: C877842	P48	Bain de minuit	200	2,00 $	400,00 $
	P37	Dentifrice doux	250	3,00 $	705,00 $

Total : 2 365,00 $

su distinguer le concept de commande et la donnée **Numéro commande**, même si aucune étiquette ou libellé sur le document ne mentionne la présence du numéro de commande. Il existe donc dans ce document une structure de données **Commande** possédant une donnée **Numéro commande** et cette structure se répète dans le document. C'est pour cette raison que dans le descriptif du document le nom **Commande** est suivi de **(*)**.

La structure **Commande** comporte de plus une référence aux produits commandés. Le modélisateur donne à cette structure le nom **Ligne commande** pour bien mettre en évidence qu'il s'agit d'un bloc de données propres à la commande. On y retrouve le numéro du produit, son nom, la quantité commandée, le prix unitaire du produit et le prix total. Cette structure se répète à l'intérieur de la structure **Commande**.

On peut noter à la lecture du descriptif du document **« Facture »** donné à la figure 1-54 que le modélisateur a pris soin de donner des noms explicites aux données de la structure **Ligne commande** car les étiquettes *Nom*, *Quantité*, *Prix* et *Total* ne sont pas clairement indicatives de la nature des données de la structure. Cette identification explicite et non ambiguë des données est d'autant justifiée que l'étiquette *Total* est aussi utilisée pour libeller ce qui est en fait le montant total de la facture (**Total facture**).

FIGURE 1-54 **Descriptif du document « Facture »**

```
Descriptif du document: Facture
Date facture
Numéro facture
Numéro client
Nom client
Adresse client
            Numéro civique
            Rue
            Ville
            État ou Province
            Pays
            Code postal
Téléphone client
Télécopieur client
Commande (*)
            Numéro Commande
            Ligne commande (*)
                        Numéro produit
                        Nom produit
                        Quantité commandée
                        Prix unitaire
                        Prix total
Total facture
```

Le descriptif d'un document est particulièrement utile au modélisateur. Non seulement permet-il une identification sans ambiguïté des données à modéliser, il reflète de plus les associations *un à plusieurs* ou *un à un* entre les données du domaine. En consultant le descriptif du document **« Facture »**, on voit nettement qu'à un numéro de facture correspondent plusieurs numéros de commandes. En vertu de la règle de description, ces données ne pourraient appartenir à la même entité mais bien à des entités qui seront liées par une association avec des multiplicités maximum valant 1 d'une part et plusieurs d'autre part. Par contre **Numéro facture** et **Numéro client**, qui sont des données de même niveau, font partie d'entités liées par une association avec des multiplicités maximum de 1 à chaque terminaison.

Pour chacune des études de cas qui suivent, un document papier est à la source d'un projet de modélisation dans un domaine particulier. Nous en produirons d'abord un descriptif selon les conventions exposées ci-dessus. Le modèle conceptuel sera ensuite élaboré sur la base du descriptif et le cas échéant d'un certain nombre de règles de gestion qui viennent compléter le cas.

CAS PORTANT SUR LES ASSOCIATIONS DE DEGRÉ SUPÉRIEUR

CAS 1-4 GESTION DE STAGE

(DP) Une institution d'enseignement professionnel offre à ses élèves des stages pratiques dans certaines disciplines. Un stage est offert à l'intérieur d'une session de stage. Le même stage peut être répété à travers plusieurs sessions. Cependant, une session de stage ne comporte qu'un seul stage avec une date de début et une date de fin.

L'élève s'inscrit donc à une session à titre de stagiaire. Une fiche de stage dûment remplie lui est remise. Cette fiche fait voir en quoi consiste le stage, notamment les modules qui le constituent. Chaque module est encadré par un ou des professeurs. La prestation du professeur comporte une date de début, une date de fin et un nombre d'heures prévues. Nous présentons ci-dessous un exemplaire de cette fiche de stage.

Fiche de stage

No stagiaire S56533 **No session** A2005

Nom, Prénom Lebel, Anne **Début session stage** 07/11/05

Date de naissance 25/03/80 **Fin session stage** 23/12/05

- [x] 001- Stage en mécanique automobile
- [ ] 002- Stage en électrotechnique
- [ ] 003- Stage en électronique
- [ ] 004- Stage en débosselage

Module no 1 — Mécanique du moteur diesel

Professeur	Date début	Date fin	Nombre d'heures
(HJ0010) Hamel, Jean	07/11/05	11/11/05	30
	14/11/05	18/11/05	40
(CS0002) Côté, Sylvie	21/11/05	25/11/05	40
(HJ0010) Hamel, Jean	28/11/05	02/12/05	20

Module no 2 — Mécanique du moteur à essence

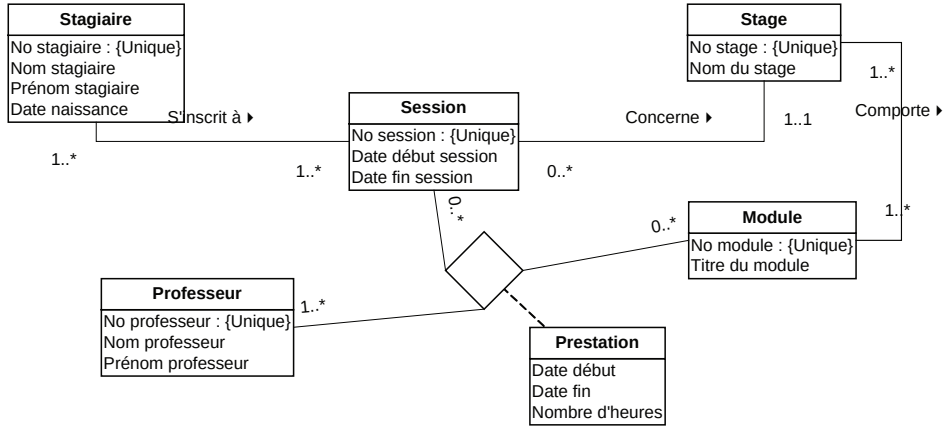
Professeur	Date début	Date fin	Nombre d'heures
(CS0002) Côté, Sylvie	05/12/05	09/12/05	30
	12/12/05	15/12/05	40
	19/12/05	23/12/05	40

Un professeur peut intervenir dans plusieurs modules du même stage. Le professeur peut par ailleurs intervenir plusieurs fois dans le même module au cours d'une session de stage. Enfin le nombre de modules varie d'un stage à l'autre selon sa nature ; il n'est pas limité à deux.

Sur la base de ces quelques règles de gestion ainsi que du contenu du document « Fiche de stage », le modélisateur a débuté son travail de modélisation en élaborant un descriptif du document qui est donné ci-après.

- **Descriptif du document** : ***Fiche de stage***
- No stagiaire
- Nom stagiaire
- Prénom stagiaire
- Date de naissance
- No stage
- Nom du stage
- No session
- Date début session
- Date fin session
- **Module(*)**
- No module
- Titre du module
- **Professeur(*)**
- No professeur
- Nom professeur
- Prénom professeur
- Date début prestation professeur
- Date fin prestation professeur
- Nombre d'heures

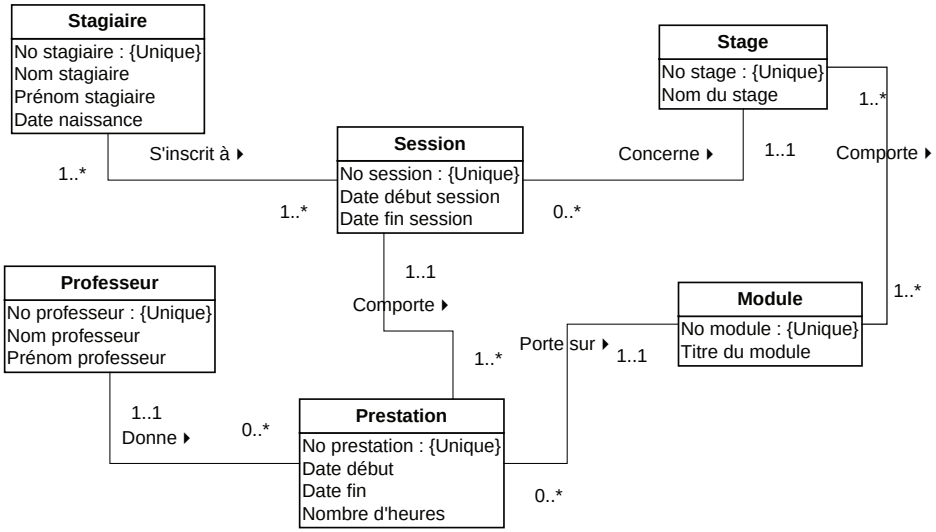
Le descriptif montre clairement qu'il existe plusieurs modules dans un stage et plusieurs professeurs interviennent dans un module. Chaque donnée du document est jugée vitale et apparaîtra donc au modèle conceptuel une seule fois sous le nom qui lui a été attribué dans le descriptif. Une première version du modèle conceptuel est exposée ici.



Le modèle fait ressortir des concepts clés dans le domaine étudié : stagiaire, stage, module, session, professeur et prestation. En vertu de la règle de construction, les dates de début et de fin de la prestation du professeur de même que le nombre d'heures de la prestation dépendent à la fois de la session, du module et du professeur. Il est donc naturel d'en faire a priori une entité d'association. Il reste à s'assurer qu'une prestation dépend FONCTIONNELLEMENT de la combinaison **No professeur-No session-No module**. Il n'en est rien, car un professeur peut donner plus d'une prestation au cours d'une même session, dans le même module.

Force est de constater que **Prestation** NE PEUT ÊTRE considérée comme une entité faible et le modélisateur doit en faire une entité avec son propre identifiant, **No prestation,** qui sera ajouté aux attributs d'une prestation.

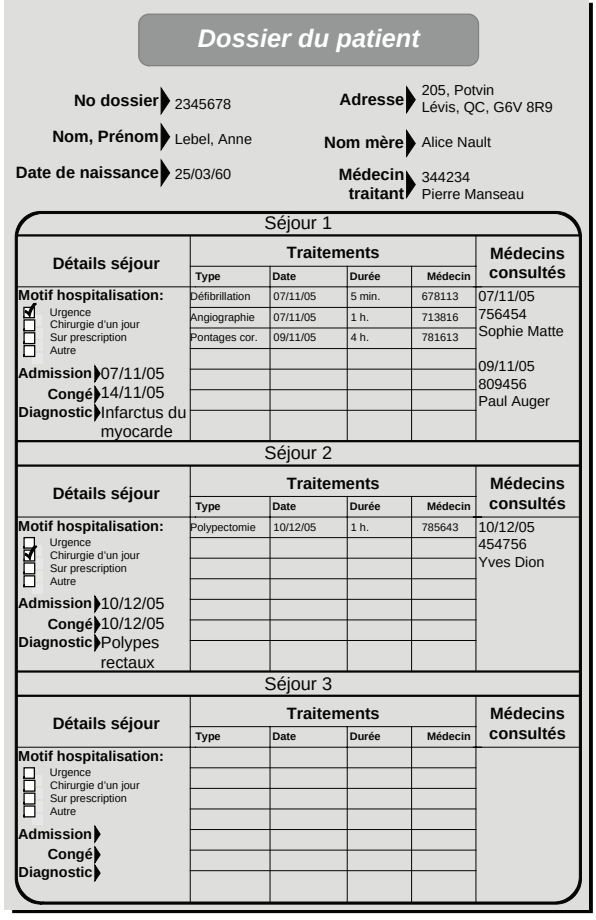
Ce cas est en fait un cas portant uniquement sur des associations binaires mais qui en apparence semblait comporter une association d'ordre supérieur. À l'analyse, puisque **Prestation** doit être considérée comme une entité forte, cette dernière doit être associée à un seul professeur, un seul module et une seule session. En conséquence le modèle initial doit être simplifié en remplaçant l'association de degré 3 par trois associations binaires.



Notons en terminant que l'association **Comporte** entre **Session** et **Prestation** ne pourrait être considérée comme une association de composition. Il n'y a pas de dépendance existentielle entre **Prestation** et **Session**. **Prestation** a son propre identifiant qui permet d'identifier chacune de ses occurrences. Il n'est pas nécessaire de faire appel en plus à l'identifiant d'une session associée pour distinguer une occurrence de prestation de toute autre.

CAS 1-5 DOSSIER PATIENT

(DP) Un petit hôpital conserve pour chacun des patients hospitalisés un dossier dans lequel sont inscrites certaines données sur la durée du séjour et les traitements reçus au cours du séjour. Le document « Dossier du patient », dont un exemplaire est reproduit ci-après, permet de consigner des informations sur trois séjours. En pratique cependant un patient peut être admis à plus de trois occasions et dans ce cas plusieurs documents seront utilisés.



Dossier du patient

No dossier 2345678
Nom, Prénom Lebel, Anne
Date de naissance 25/03/60

Adresse 205, Potvin
Lévis, QC, G6V 8R9
Nom mère Alice Nault
Médecin traitant 344234
Pierre Manseau

Séjour 1

Détails séjour	Traitements				Médecins consultés
	Type	Date	Durée	Médecin	
Motif hospitalisation:	Défibrillation	07/11/05	5 min.	678113	07/11/05 756454 Sophie Matte
☑ Urgence ☐ Chirurgie d'un jour ☐ Sur prescription ☐ Autre	Angiographie	07/11/05	1 h.	713816	
	Pontages cor.	09/11/05	4 h.	781613	
Admission 07/11/05 **Congé** 14/11/05 **Diagnostic** Infarctus du myocarde					09/11/05 809456 Paul Auger

Séjour 2

Détails séjour	Traitements				Médecins consultés
	Type	Date	Durée	Médecin	
Motif hospitalisation:	Polypectomie	10/12/05	1 h.	785643	10/12/05 454756 Yves Dion
☐ Urgence ☑ Chirurgie d'un jour ☐ Sur prescription ☐ Autre					
Admission 10/12/05 **Congé** 10/12/05 **Diagnostic** Polypes rectaux					

Séjour 3

Détails séjour	Traitements				Médecins consultés
	Type	Date	Durée	Médecin	
Motif hospitalisation:					
☐ Urgence ☐ Chirurgie d'un jour ☐ Sur prescription ☐ Autre					
Admission **Congé** **Diagnostic**					

Le modélisateur a dressé un descriptif du document qui montre clairement qu'un médecin peut intervenir auprès du patient à titre de médecin traitant, c'est-à-dire en médecin de famille, et ce médecin intervient seul à ce titre pour un patient. Par ailleurs un médecin peut intervenir à titre de médecin consulté dans le cadre d'un séjour du patient. Plusieurs médecins peuvent intervenir à ce titre lors d'un séjour du patient. Un médecin peut enfin appliquer un traitement au patient durant un de ses séjours. Là encore plusieurs médecins peuvent agir à ce titre pour différents traitements effectués lors d'un même séjour.

Le descriptif illustre bien les liens de multiplicité suivants : un patient réfère à un seul médecin traitant ; un patient réfère à plusieurs séjours ; un séjour réfère à plusieurs traitements ; un traitement réfère à un seul médecin ; un séjour réfère à plusieurs médecins consultés.

Descriptif du document: ***Dossier du patient***

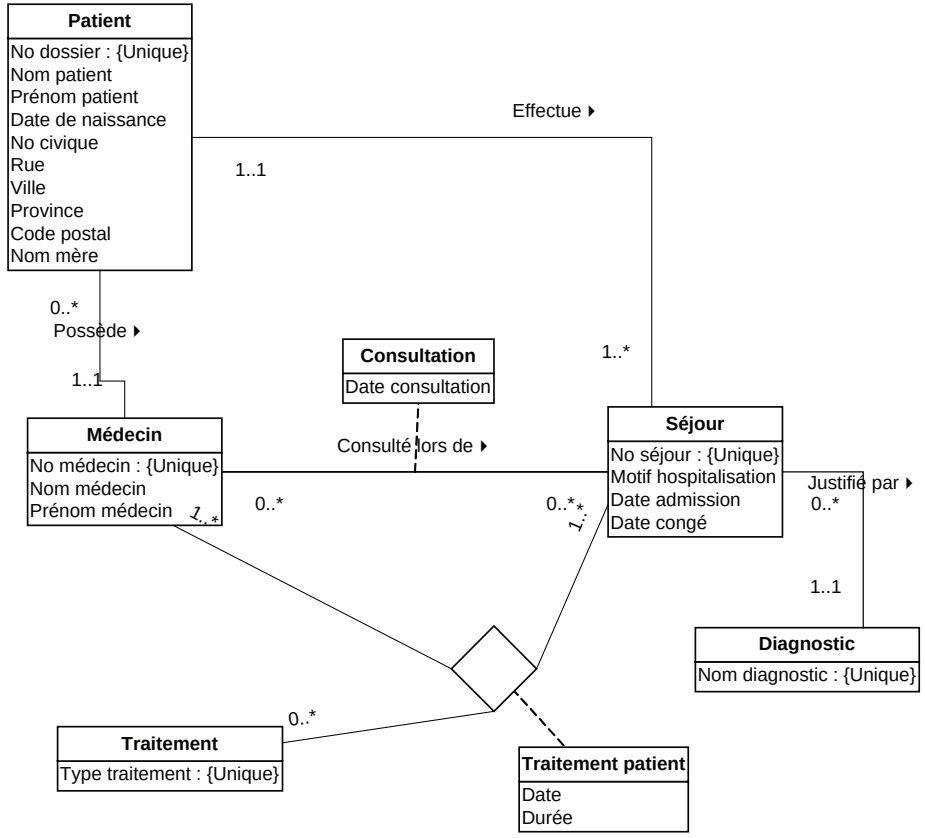
- No dossier
- Nom patient
- Prénom patient
- Date de naissance
- **Adresse patient**
  - No civique
  - Rue
  - Ville
  - Province
  - Code postal
- Nom mère
- **Médecin traitant**
  - No médecin
  - Nom médecin
  - Prénom médecin
- **Séjour à l'hôpital(*)**
  - Motif hospitalisation
  - Date admission
  - Date congé
  - Nom diagnostic
  - **Traitement(*)**
    - Type traitement
    - Date
    - Durée
    - No médecin
  - **Médecin consulté(*)**
    - Nom médecin
    - Prénom médecin
    - No médecin
    - Date consultation

Le modélisateur devra éviter de confondre l'entité **Médecin** et le rôle que joue un médecin auprès d'un patient. Seule l'entité **Médecin** doit refléter les attributs de tous les médecins quel que soit leur rôle : **No médecin**, **Nom médecin**, **Prénom médecin**. Leur rôle devra se refléter à travers des associations entre l'entité **Médecin** et d'autres entités pertinentes telles que **Patient** ou **Séjour**.

Deux règles de gestion très importantes, qui ne peuvent être extrapolées du descriptif du document, devront être prises en compte par le modélisateur dans ce cas :

- **règle 1** − un type de traitement donné ne peut être appliqué plus d'une fois par le même médecin lors d'un séjour d'hospitalisation du patient ;
- **règle 2** − un médecin fait au plus une consultation lors d'un séjour du patient.

Comme nous le verrons, ces règles de gestion ont un impact déterminant sur le modèle conceptuel produit par le modélisateur et donné ci-après.



Le modélisateur a jugé pertinent de faire de **Diagnostic** et **Traitement** des entités. La raison en est simple : dans le milieu hospitalier, il s'agit de concepts fondamentaux qui ont une existence propre. Bien que le modèle ne fait voir qu'un seul attribut pour chacun, il y lieu de croire qu'un spécialiste du domaine pourrait en identifier facilement plusieurs autres.

La règle de gestion numéro 2 voulant qu'un médecin effectue au plus une consultation dans le cadre d'un séjour du patient justifie la présence d'une entité d'association **Consultation** dont l'attribut **Date consultation** dépend à la fois de **Médecin** et de **Séjour** et cette dépendance est fonctionnelle. **Consultation** est nettement une entité faible, dont l'identifiant implicite est la combinaison des deux attributs : **No médecin-No séjour**.

La règle de gestion numéro 1 voulant qu'un médecin ne pratique jamais plus d'une fois le même traitement dans le cadre d'un séjour du patient, justifie la présence d'une entité d'association **Traitement patient** dont les attributs **Date** et **Durée** dépendent à la fois de **Médecin** de **Séjour** et de **Traitement**. Cette dépendance est fonctionnelle. **Traitement patient** est en effet une entité faible, dont l'identifiant implicite est la combinaison de trois attributs : **No médecin-No séjour-Type de traitement**.

Comment sont justifiées les multiplicités sur l'association de degré 3 **Traitement patient** ?

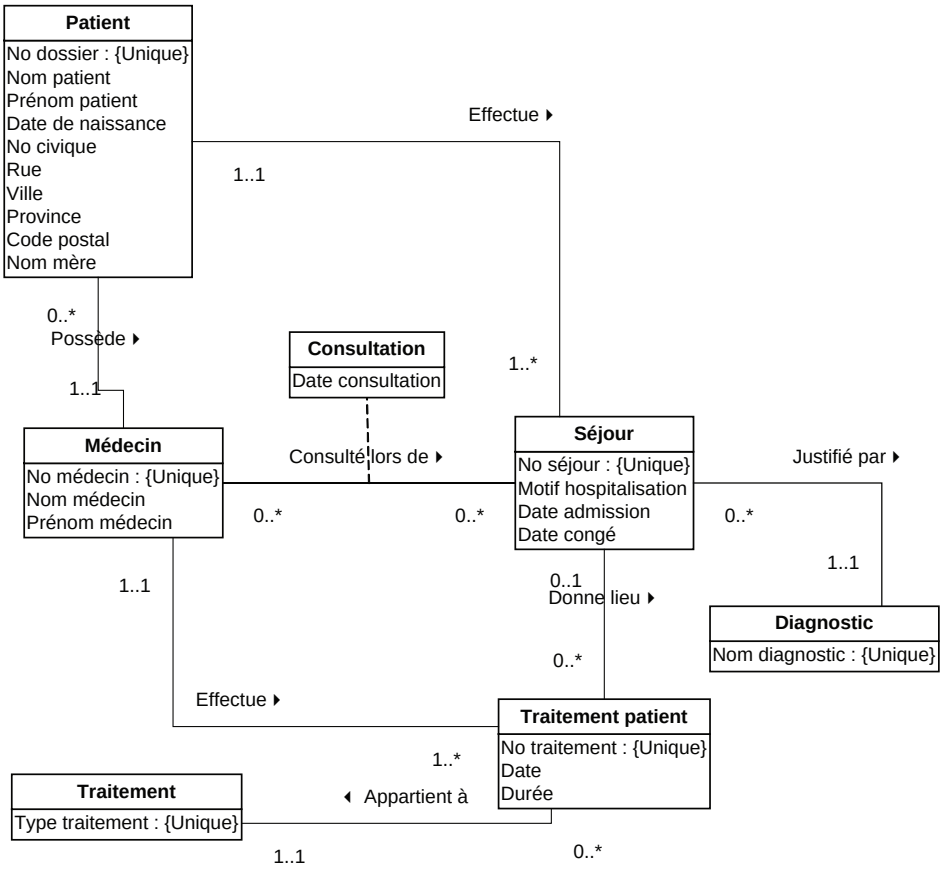
Côté **Médecin** : combien de médecins sont impliqués dans une association **Séjour-Traitement** ? Un ou plusieurs (1..*). Le même traitement peut être répété au cours du même séjour du patient pour peu qu'il le soit par des médecins différents comme le prescrit la règle 1.

Côté **Séjour** : combien de séjours sont impliqués dans une association **Médecin-Traitement** ? Au moins 1 (1..*), car si un type de traitement est donné par un médecin il doit l'être au moins au cours du séjour d'un patient et éventuellement lors de plusieurs séjours différents.

Côté **Traitement** : combien de traitements sont liés à une association **Médecin-Séjour** ? Un nombre indéterminé (0..*), car un médecin peut être lié à un séjour du patient sans lui avoir donné de traitement (pour une consultation par exemple), mais pourrait donner plusieurs types de traitements différents lors de ce séjour.

Comme ce cas le démontre, la présence dans un domaine d'activités de règles de gestion précises influe considérablement sur le modèle qui en découle. Considérons maintenant que la règle 1 est relaxée. Dès lors une occurrence de **Traitement patient** ne peut dépendre fonctionnellement de la combinaison des identifiants **No médecin-No séjour-Type de traitement**. **Traitement patient** devra être considéré comme une entité forte et devra en conséquence disposer de son propre identifiant : **No traitement**. Notons au passage que dans le modèle révisé, suite à la relaxation de la règle 1, on observe sur l'association **Donne lieu** des multiplicités 0..1 côté **Séjour**. C'est qu'en toute généralité, un traitement

effectué sur un patient ne s'inscrit pas exclusivement dans le cadre d'un séjour à l'hôpital. Il peut se faire par exemple en cabinet (0). S'il est effectué à la suite d'une hospitalisation, le traitement s'inscrit alors dans au plus un séjour (1).



CAS PORTANT SUR LES ASSOCIATIONS SPÉCIALISÉES

CAS 1-6 MEMBRE CLUB DE TENNIS

(DP) Le club de tennis Chaudière inscrit sur une fiche les résultats de la participation d'un de ses membres à divers tournois, que le tournoi soit organisé par le club ou non.

Comme l'illustre un exemplaire de cette fiche, chaque tournoi possède un nom et une date de début. Les matchs auxquels le membre a participé s'inscrivent dans un tournoi et le match possède une date, la nature de la finale à laquelle le membre a participé et un type.

Club de tennis Chaudière

No membre 16725

Nom, Prénom Lebeau, Pierre

☑ Homme
☐ Femme

Tournois	Matchs du tournoi			
	Date	**Nature match**	**Type match**	**Gagné?**
Date début: 11/07/05 **Nom:** Challenge Lévis 2005	12/07/05 13/07/05 15/07/05	½ finale finale ¼ finale	simple homme simple homme double homme	✓ ✓
Date début: 1/08/05 **Nom:** Challenge Lauberivière 2005	02/08/05 02/08/05 03/08/05	¼ finale ½ finale finale	double mixte double mixte double mixte	✓ ✓ ✓

Le descriptif du document montre nettement les liens de multiplicité entre les données : un membre participe à PLUSIEURS tournois, chaque tournoi est constitué de PLUSIEURS matchs joués par le membre et chaque match est gagné ou non par ce dernier.

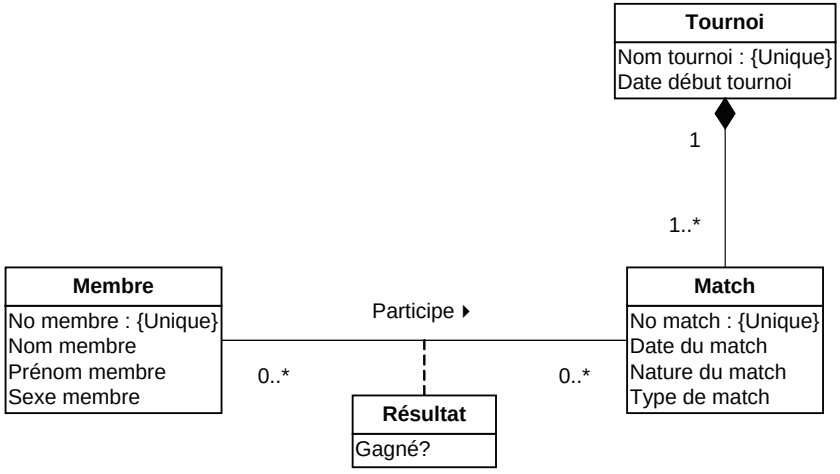
Le modélisateur a bien mis en évidence dans le descriptif du document les valeurs que peuvent prendre certaines données en les plaçant entre parenthèses à la suite du nom de la donnée. C'est le cas des données **Nature du match** et **Type de match**. Il ne s'agit pas d'une exigence absolue que de faire état dans le descriptif d'une liste des valeurs que peut prendre une donnée. Le modélisateur en fait usage ici pour bien faire la distinction entre une donnée et les valeurs de cette donnée qui peuvent apparaître dans un exemplaire du document.

Descriptif du document : ***Club de tennis Chaudière***

- No membre
- Sexe joueur
- Nom membre
- Prénom membre
- **Tournoi(*)**
- Nom du tournoi
- Date début tournoi
- **Match(*)**
- Date du match
- Nature du match -(32e de finale,…, ½ finale, finale)
- Type de match -(simple homme, double mixte, etc…)
- Gagné?

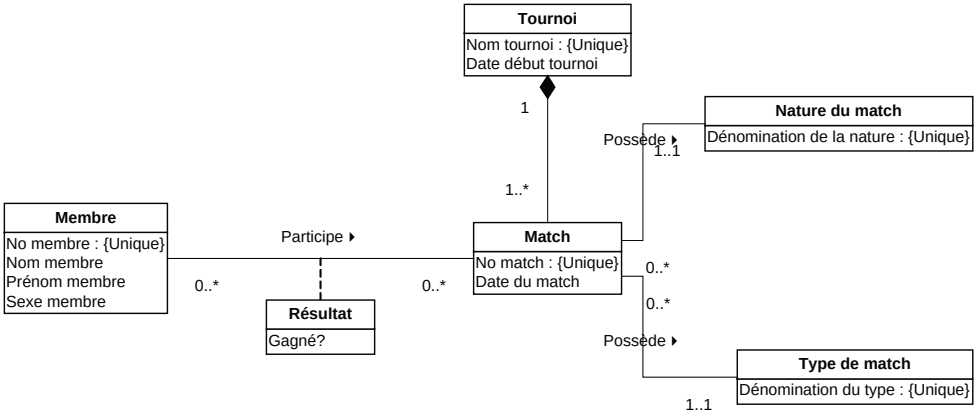
Le modèle conceptuel réalisé par le modélisateur montre une composition. Un tournoi est constitué de plusieurs matchs et les matchs ne peuvent exister indépendamment d'un tournoi. La dépendance d'un match envers le tournoi dans lequel il s'inscrit est totale. Cela implique sur le plan sémantique que la suppression d'un tournoi entraîne la suppression des matchs qui le constituent. La suppression du tout entraîne la disparition de chacune de ses parties.

Par convention l'entité composant (ici **Match**) ne montre que l'identifiant qui permet de distinguer chacune des occurrences à l'intérieur du composite (**No match**). Cela signifie que l'identifiant réel de **Match** est ici la combinaison **Nom tournoi-No match**, car un même numéro de match peut exister dans plusieurs tournois. Par ailleurs, puisque le **Résultat** d'un match dépend fonctionnellement de **Membre** et de **Match**, cette entité faible est justifiée.



L'entité **Match** comporte deux attributs dont le type est énuméré. En fait les valeurs possibles pour les attributs **Nature du match** et **Type de match** sont connues a priori et leur nombre est fini. Bien que non nécessaire sur le plan théorique, il y a en pratique de nombreux avantages à faire des entités distinctes avec les attributs de type énuméré.

Ces entités sont considérées comme des ENTITÉS DE DESCRIPTION. Elles permettent de qualifier, de classifier ou de catégoriser d'autres entités. On peut par exemple ajouter au modèle précédent deux entités, **Nature du match** et **Type de match** , qui seront associées à l'entité **Match** avec des multiplicités maximales de 1 et plusieurs, pour permettre de qualifier les matchs. Les attributs correspondants sont par ailleurs retirés de **Match**.

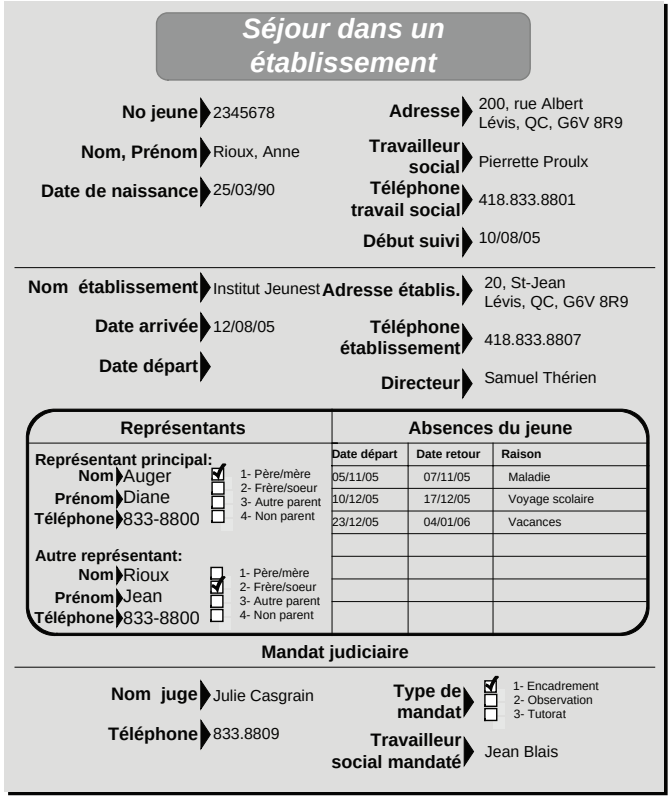


Entité de description ▸ Entité comportant généralement un seul attribut, souvent de type énuméré, permettant en l'associant à une autre entité avec des multiplicités maximales de 1 et plusieurs, de décrire cette dernière et plus spécifiquement de la classifier ou de la catégoriser (*Description entity*).

Les avantages de cette classe d'entités ne sont cependant pas immédiats. Comme nous le verrons au chapitre 2, leur présence a un impact majeur sur la qualité du modèle logique de données qui découle du modèle conceptuel. Nous aurons le loisir d'y revenir lorsque nous traiterons du passage du modèle conceptuel au modèle logique.

CAS 1-7 SÉJOUR DANS UN ÉTABLISSEMENT JEUNESSE

(DP) Un organisme d'aide à la jeunesse consigne dans un dossier des données relatives aux séjours effectués par un jeune dans divers établissements dont la vocation est d'héberger temporairement des jeunes en difficulté. Chaque séjour d'un jeune dans ce type d'établissement fait l'objet d'une inscription complétée par le biais du formulaire « Séjour dans un établissement ».



Séjour dans un établissement

No jeune 2345678
Nom, Prénom Rioux, Anne
Date de naissance 25/03/90
Adresse 200, rue Albert Lévis, QC, G6V 8R9
Travailleur social Pierrette Proulx
Téléphone travail social 418.833.8801
Début suivi 10/08/05

Nom établissement Institut Jeunest
Date arrivée 12/08/05
Date départ
Adresse établis. 20, St-Jean Lévis, QC, G6V 8R9
Téléphone établissement 418.833.8807
Directeur Samuel Thérien

Représentants

Représentant principal:
Nom Auger
Prénom Diane
Téléphone 833-8800
☑ 1- Père/mère
☐ 2- Frère/soeur
☐ 3- Autre parent
☐ 4- Non parent

Autre représentant:
Nom Rioux
Prénom Jean
Téléphone 833-8800
☐ 1- Père/mère
☑ 2- Frère/soeur
☐ 3- Autre parent
☐ 4- Non parent

Absences du jeune

Date départ	Date retour	Raison
05/11/05	07/11/05	Maladie
10/12/05	17/12/05	Voyage scolaire
23/12/05	04/01/06	Vacances

Mandat judiciaire

Nom juge Julie Casgrain
Téléphone 833.8809
Type de mandat ☑ 1- Encadrement ☐ 2- Observation ☐ 3- Tutorat
Travailleur social mandaté Jean Blais

Le document comporte des données sur le jeune, sur l'établissement et sur le séjour. En ce qui a trait au séjour, on note la date d'arrivée et la date de départ ainsi que les périodes d'absence du jeune au cours d'un séjour. Tout jeune doit avoir un représentant principal et le cas échéant un deuxième représentant. Si le séjour découle d'un mandat judiciaire, le juge qui a accordé le mandat et le travailleur social mandaté sont identifiés sur le document.

Descriptif du document: ***Séjour dans un établissement***

- No jeune
- Nom jeune
- Prénom jeune
- Date de naissance
- **Adresse jeune**
  - No civique
  - Rue
  - Ville
  - Province
  - Code postal
- Nom travailleur
- Prénom travailleur
- Téléphone travailleur
- Début suivi
- Nom établissement
- **Adresse établissement**
  - No civique
  - Rue
  - Ville
  - Province
  - Code postal
- Téléphone établissement
- Directeur
- Date d'arrivée
- Date de départ
- **Représentant(*)**
  - Nom représentant
  - Prénom représentant
  - Téléphone représentant
  - Code de parenté
  - Représentant principal?
- **Absence(*)**
  - Date de début
  - Date de retour
  - Raison
- **Mandat**
  - Nom juge
  - Prénom juge
  - Téléphone juge
  - Type de mandat
  - Nom travailleur
  - Prénom travailleur

Le modélisateur a su reconnaître dans le document une structure qui se répète, **Représentant**, même si cette répétition est au nombre de deux. Elle est bien justifiée car il s'agit des mêmes données qui se répètent. On y distingue le représentant principal d'un second représentant.

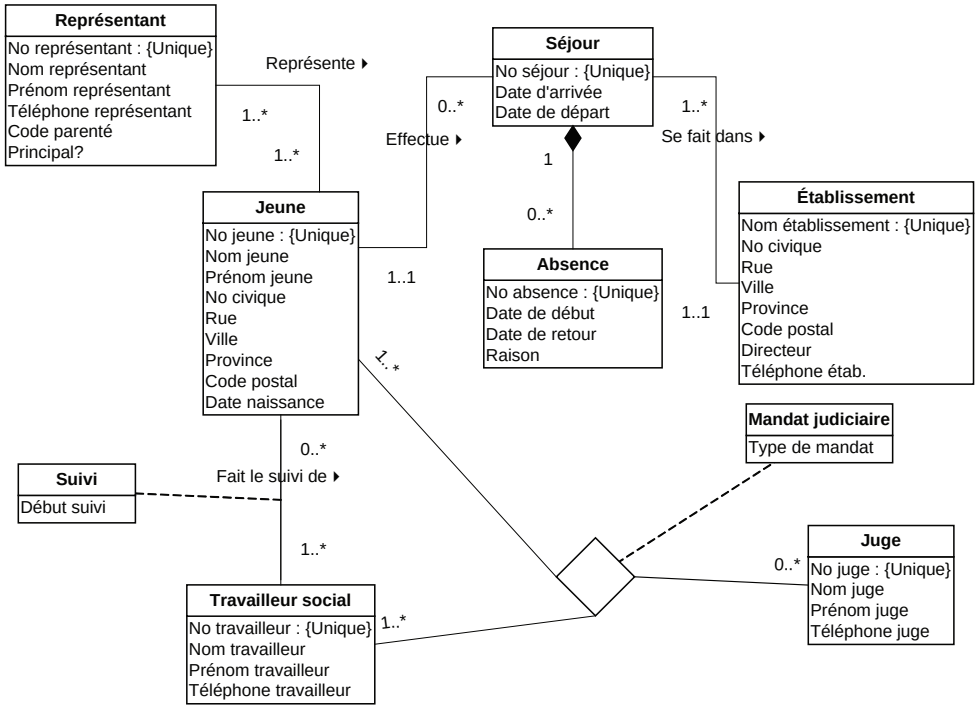
Une structure **Mandat** est aussi présente, permettant de mettre en évidence les données relatives à un mandat judiciaire qui aurait donné lieu à un séjour dans l'établissement. Il ne s'agit point d'une structure répétitive.

Le modèle conceptuel qui en découle repose sur certaines règles de gestion qui permettent de justifier la présence des deux entités d'association. Primo, bien que plusieurs travailleurs sociaux aient assuré à un moment ou l'autre le suivi d'un jeune, un travailleur social n'assure jamais deux fois le suivi du même jeune. Cette règle offre la garantie que la combinaison **No jeune-No travailleur** ne produit jamais de doublon et permet de considérer **Suivi** comme une entité faible, donc valable comme entité d'association. Secundo, deux mandats judiciaires ne peuvent mettre en cause à la fois le même jeune, le même juge et le même travailleur. Là encore cette règle permet de considérer **Mandat judiciaire** comme une entité faible, acceptable comme entité d'association avec un identifiant implicite composé de **No jeune**, **No juge** et **No travailleur**.

Les connectivités de l'association **Mandat Judiciaire** se justifient ainsi :

1- Côté **Juge** ; un jeune et un travailleur social pourraient ne pas être associés à un juge, si aucun mandat n'existe à son propos, ou au contraire à plusieurs car un jeune peut faire l'objet de plusieurs mandats (0..*)

2- Côté **Jeune** ; un travailleur social et un juge, lorsqu'associés, le sont à au moins un jeune ou à plusieurs le cas échéant (1..*)



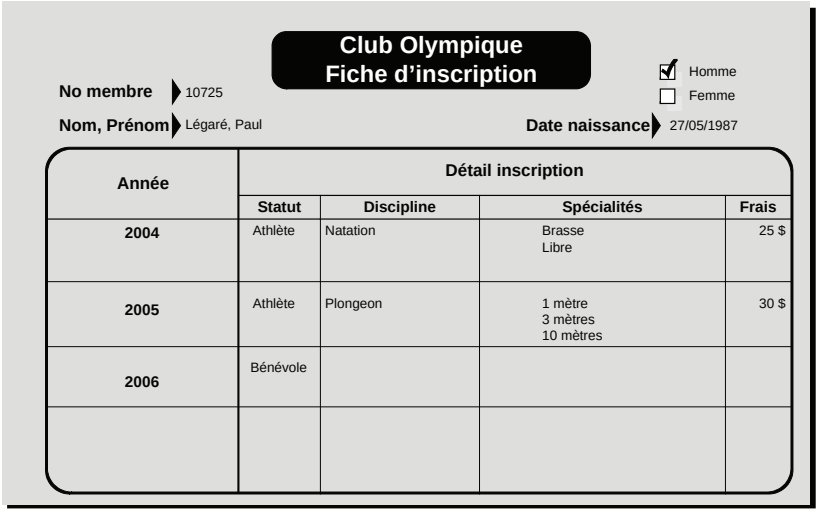
3- Côté **Travailleur** ; un jeune et un juge lorsqu'associés, le sont à au moins un travailleur social ou à plusieurs le cas échéant (1..*) si le juge accorde plusieurs mandats au jeune

Une entité composite appelée **Séjour** se justifie ici par l'existence de dépendance totale d'une absence du jeune en regard d'un séjour de ce dernier dans un établissement. L'identifiant réel de l'entité **Absence** est la combinaison de **No séjour-No absence**. La composition montre une multiplicité minimale 0 côté **Absence**. Autrement dit un séjour est composé d'aucune ou de plusieurs périodes d'absence mais une absence ne peut exister indépendamment d'un séjour.

CAS 1-8 CLUB D'ATHLÉTISME

Un club d'athéisme gère les inscriptions de ses membres par le biais d'une fiche où annuellement le membre choisit à quel titre il complète son inscription. Les règles de gestion en ce qui concerne les inscriptions sont les suivantes :

1. Lorsque le membre s'inscrit à une année donnée, il peut choisir de le faire à titre de bénévole ou d'athlète (le statut)
2. S'il s'inscrit à titre d'athlète, il doit indiquer une ou plusieurs spécialités qu'il souhaite pratiquer. Chaque spécialité doit être rattachée à une discipline. À titre de bénévole aucune spécialité n'est choisie
3. À titre d'athlète, il doit payer des frais d'inscription, aucun frais n'est assumé par un bénévole
4. Le sexe et la date de naissance sont des données vitales pour un athlète, pas pour un bénévole
5. Un membre ne peut s'inscrire qu'à un seul titre au cours d'une année, mais peut changer de statut d'une année à l'autre.



Club Olympique
Fiche d'inscription

No membre 10725

☑ Homme
☐ Femme

Nom, Prénom Légaré, Paul

Date naissance 27/05/1987

Année	Détail inscription			
	Statut	Discipline	Spécialités	Frais
2004	Athlète	Natation	Brasse Libre	25 $
2005	Athlète	Plongeon	1 mètre 3 mètres 10 mètres	30 $
2006	Bénévole			

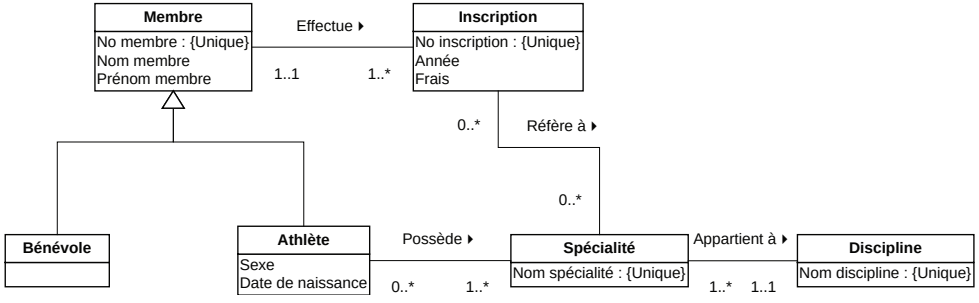
Le descriptif de la fiche est modélisé ainsi :

Descriptif du document : ***Club Olympique***

- No membre
- Sexe joueur
- Nom membre
- Prénom membre
- Date naissance
- **Inscription(*)**
- Année
- Statut
- Frais
- **Spécialité(*)**
- Nom spécialité
- Discipline

Le descriptif établit nettement qu'un membre peut faire plusieurs inscriptions au cours des ans et que chaque inscription fait référence le cas échéant à des spécialités.

Cependant, seul le modèle conceptuel peut véhiculer les autres règles de gestion, notamment les statuts mutuellement exclusifs que peut posséder un membre. Une association d'héritage permet de montrer qu'un membre peut être soit un **Bénévole**, soit un **Athlète** mais que seul l'athlète peut posséder des spécialités d'une part et que d'autre part la Date de naissance et le Sexe sont des données vitales, pertinentes uniquement à l'athlète. Une association entre l'entité **Inscription** et l'entité **Spécialité** est requise pour établir le statut du membre durant une année donnée et pour savoir à quelles spécialités l'athlète est rattaché cette année là.



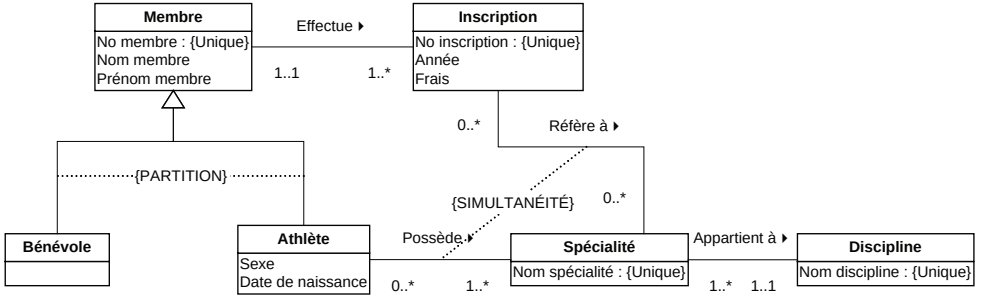
CAS PORTANT SUR LES CONTRAINTES INTER-ASSOCIATIONS

CAS 1-9 CLUB D'ATHLÉTISME

Le cas précédent peut facilement être complété par des contraintes inter-associations. Puisqu'on on y note la présence d'une association d'héritage comportant deux sous-types, il est tout à fait approprié de préciser si un sous-type exclut l'autre d'une part et d'autre part, si une occurrence du supertype peut exister sans appartenir à un des sous-types. Une contrainte de PARTITION spécifie deux choses :

1. un statut (**Bénévole** ou **Athlète**) exclut l'autre. En effet un membre ne peut être à la fois **Bénévole** et **Athlète**
2. un membre appartient obligatoirement à un des sous-types. Autrement dit, une occurrence du supertype **Membre** ne peut exister à moins d'appartenir à un des sous-types.

On peut considérer par ailleurs que si une occurrence de **Spécialité** est liée à une occurrence de **Athlète**, il y a forcément une liaison de cette spécialité à une occurrence de **Inscription** car c'est par le biais d'une inscription que se fait le choix de la spécialité d'un athlète.



Ces deux dernières contraintes, bien que non requises pour compléter le modèle conceptuel de données, apportent une meilleure illustration des règles de gestion du club d'athlétisme.

ÉVOLUTION DES NOTATIONS POUR LA MODÉLISATION CONCEPTUELLE DES DONNÉES

Dès le début des années 1970, considérant l'importance grandissante des bases de données pour la gestion des données, plusieurs équipes de chercheurs se sont intéressées au développement d'une notation graphique pour modéliser les données sur le plan conceptuel.

On attribue généralement à Peter Chen la paternité du formalisme entité-association. Le formalisme et la notation d'origine ont été décrits dans un célèbre papier daté de 1976 [CHE 76], un des plus cités de la littérature portant sur les bases de données. Parallèlement, en France, une équipe dirigée par H. Tardieu élaborait un formalisme du même type appelé *Formalisme individuel*. Les échanges entre les chercheurs des deux côtés de l'Atlantique ont donné naissance à ce qu'on appelle désormais en français le formalisme *entité-association* ou *entity-relationship* chez les Anglo-Saxons.

Malheureusement, après plus de 30 ans d'évolution, aucun consensus ne s'est établi quant à la notation la plus appropriée pour exprimer le formalisme. Bien que UML semble vouloir prendre le pas sur les deux autres notations, la notation de Chen et ses variantes ainsi que la notation Merise demeurent largement utilisées et de nombreux outils y font appel.

Dans cette dernière section du chapitre 1, nous proposons une brève description de ces notations ainsi que de leur évolution dans le temps.

Notation de Chen

À l'origine, la notation proposée par Chen faisait appel au rectangle pour représenter les entités types, au losange pour une association type et à l'ovale pour un attribut. Seules les multiplicités maximales sont présentes sur les arcs reliant une entité et une association. Elle est soit 1 ou plusieurs (N ou M). Les contraintes de multiplicité sont appelées des cardinalités (*cardinality*).

La figure 1-55 montre un modèle conceptuel selon la notation originale de Chen. Il illustre les concepts de la gestion de projet. On note que chaque attribut est représenté par un ovale. Un des attributs est souligné, ce qui en fait un identifiant de l'entité (*Key attribute*). De plus la position des multiplicités est conforme à la représentation UML quant aux associations binaires : pour une lecture de l'association, la multiplicité présente du côté de l'entité d'arrivée indique le nombre d'occurrences de cette dernière qui sont liées à une occurrence de l'entité de départ.

Très tôt des extensions à la notation ont été proposées par des auteurs américains notamment pour alléger la notation et permettre d'exprimer des multiplicités minimales. C'est ainsi que les multiplicités minimales et maximales ont pu être exprimées grâce à une représentation symbolique à l'extrémité d'un arc. Cette forme de représentation est dite en *ergot de coq*. Le tableau 1-4 montre les quatre combinaisons de multiplicité adoptées en UML et leur forme correspondante en ergot de coq. Une variante de la notation de Chen fait appel à une représentation numérique des multiplicités à la manière UML mais elle se distingue par l'emploi de la virgule comme séparateur des multiplicités.

Dans l'évolution finale de la notation de Chen, il fut proposé que les attributs soient intégrés à leur entité ou à leur association, de manière à réduire considérablement la taille du modèle.

FIGURE 1-55 **Notation de Chen pour modéliser la gestion de projet**

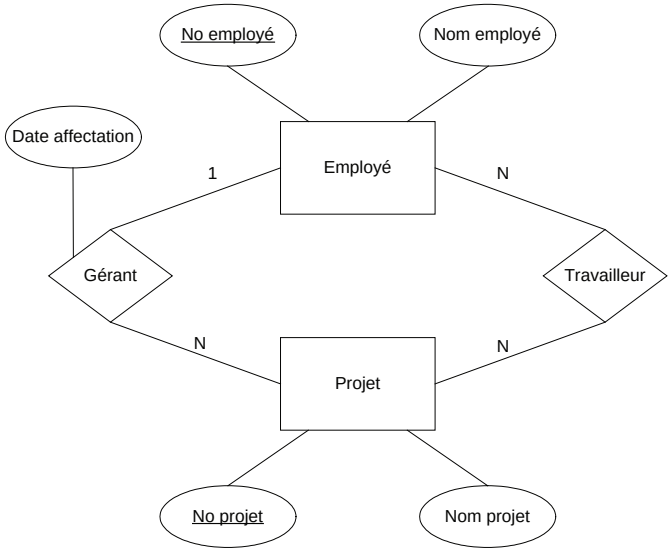
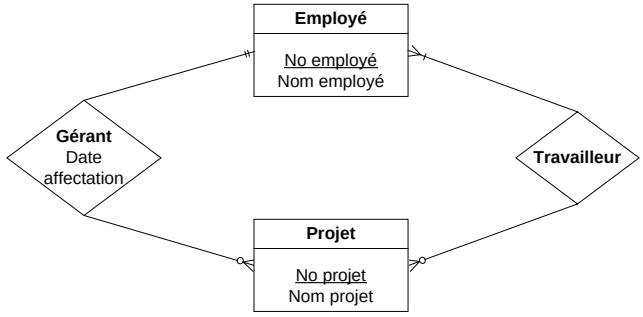


TABLEAU 1-4 **Codification graphique des multiplicités en ergot de coq**

Multiplicité UML	Signification	Ergot de coq	Représentation numérique
`0..1`	Au plus un		`(0,1)`
`1..1 (ou 1)`	Un seul		`(1,1)`
`0..* (ou *)`	Un nombre indéterminé		`(0,N)`
`1..*`	Au moins un		`(1,N)`

La figure 1-56 offre une version allégée du modèle 1-55 faisant appel aux extensions décrites ci-dessus avec des multiplicités en ergot de coq. Le même modèle est repris à la figure 1-57 en faisant appel cette fois à des multiplicités exprimées numériquement.

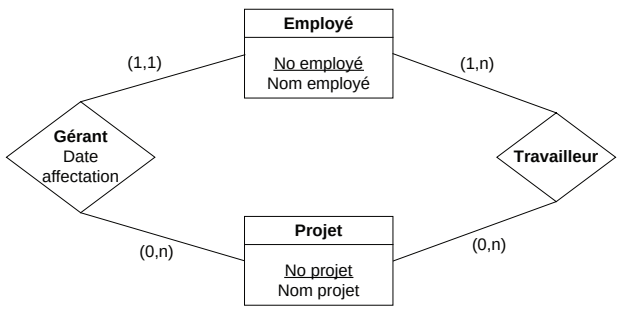
FIGURE 1-56 **Notation de Chen allégée pour modéliser la gestion de projet**



La notion *entité d'association* n'existe pas dans la notation de Chen, ni pour les associations binaires, ni pour celles de degré supérieur. Si une association est dotée d'attributs, ceux-ci sont inscrits dans l'association comme à la figure 1-56 où l'attribut **Date affectation** appartient à l'association **Gérant**. La lecture des multiplicités d'une association de degré supérieur est conforme à UML. Ce sont d'ailleurs les concepteurs de UML qui s'en sont inspirés. Pour une association de degré *n*, les multiplicités présentes du côté d'une entité représentent le nombre d'occurrences de cette entité qui peuvent être associés à un tuple composé d'une occurrence de chacune des *n–1* autres entités.

La notation de Chen ne permet pas de formuler des contraintes inter-associations, ni des contraintes spécialisées. De plus, comme en UML, seules des entités peuvent être liées par une association, il est sémantiquement incorrect de lier deux associations.

FIGURE 1-57 **Notation de Chen avec multiplicités numériques**



Notation de Merise

Merise, tout comme UML, est bien plus qu'une notation. Il s'agit – dans sa livrée Merise/2 – d'un ensemble de méthodes, de formalismes, de notations et d'outils pour le développement de systèmes d'information. À l'instar de UML, Merise/2 permet de concevoir une base de données en faisant notamment appel au formalisme entité-association.

La notation Merise adoptée pour élaborer un modèle conceptuel avec le formalisme entité-association se distingue de la notation de Chen principalement par la richesse de ses contraintes sémantiques. Ces contraintes sémantiques sont surtout exprimées par des contraintes inter-associations et des contraintes spécialisées comme en UML. Merise a servi d'inspiration aux auteurs de UML sur ce plan.

Une autre différence, cette fois mineure, est d'ordre cosmétique : une association est représentée par un ovale plutôt que par un losange.

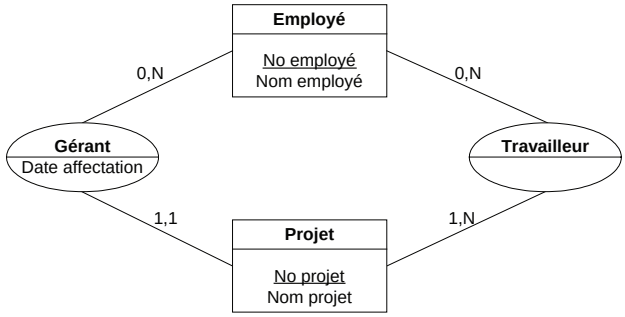
Tout comme dans la notation de Chen, les attributs des entités et des associations sont intégrés à ces symboles et les entités d'association n'ont pas de raison d'être. L'identifiant d'une entité est souligné et il peut être composé. Il est sémantiquement incorrect de lier des associations et celles-ci ne disposent pas d'identifiant car il est implicite comme pour la notation de Chen.

La différence la plus fondamentale entre Merise et UML (et un des prédécesseurs de UML, la notation de Chen) réside dans la logique d'expression des multiplicités. Les concepteurs de Merise ont voulu donner plus de cohérence à la logique de formulation des multiplicités avec pour conséquence qu'elle se distingue de manière fondamentale de celle retenue par Chen et par les concepteurs de UML qui ont endossé les prescriptions de Chen.

Avec la notation Merise, que ce soit pour une association binaire ou une association de degré supérieur, la multiplicité placée du côté d'une entité l'est toujours dans le contexte où cette entité est l'entité de départ pour la lecture de l'association.

Pour une association binaire : la multiplicité placée à côté de l'entité de départ stipule le nombre d'occurrences de l'entité d'arrivée qui participe à l'association avec cette première. Cela représente une inversion complète de toutes les multiplicités sur les associations binaires par rapport à un modèle formulé avec la notation de Chen. La figure 1-58 reprend le modèle 1-57 exprimé en notation Merise. Prière de noter l'inversion des multiplicités alors que les autres aspects du modèle sont similaires, les losanges faisant place aux ovales pour représenter les associations.

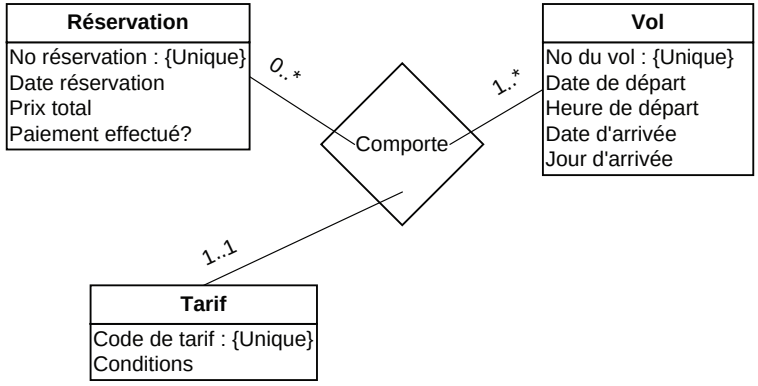
FIGURE 1-58 **Notation Merise pour modéliser la gestion de projet**



Quant à une association de degré supérieur ***n***: la multiplicité placée du côté de l'entité de départ stipule le nombre de tuples, formés d'une occurrence de chacune des ***n–1*** autres entités, qui peuvent participer à une association avec cette première.

Considérons le modèle 1-59 repris de la figure 1-25 et exprimé dans la notation UML. Nous invitons le lecteur à relire les exigences du problème et les justifications concernant les multiplicités.

FIGURE 1-59 **Réservation auprès d'une société aérienne exprimée en UML**



En vertu de la logique de formulation des multiplicités en Merise, nous allons devoir établir pour chaque entité de l'association, combien de couples formés d'occurrences des deux autres entités peuvent être associés à cette première.

Considérons chaque entité de l'association à tour de rôle.

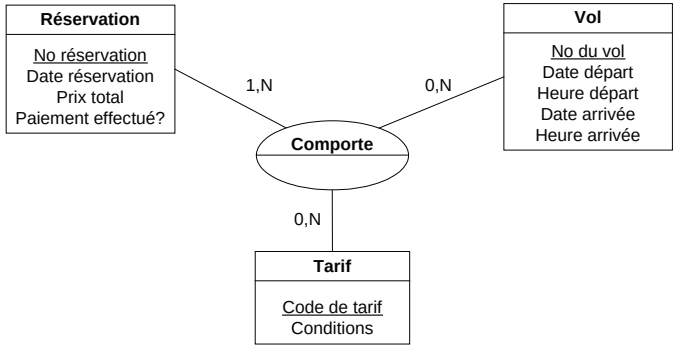
Multiplicité du côté **Réservation** : Combien de couples **Tarif-Vol** peuvent être associés à une occurrence de **Réservation** ? Une réservation comporte au moins un vol donc au moins un couple **Tarif-Vol**. Un ou plusieurs, donc multiplicité 1,N respectant en cela la syntaxe des multiplicités en Merise.

Multiplicité du côté **Tarif** : Combien de couples **Réservation-Vol** peuvent être associés à une occurrence de **Tarif** ? Un tarif peut ne pas s'appliquer aux réservations de vols effectuées à un moment donné mais pourrait s'appliquer à plusieurs. Zéro ou plusieurs, donc multiplicité 0,N respectant en cela la syntaxe des multiplicités en Merise.

Multiplicité du côté **Vol** : Combien de couples **Réservation-Tarif** vols peuvent être associés à une occurrence de **Vol** ? Un vol pourrait ne pas avoir de réservations mais éventuellement plusieurs réservations avec tarifs différents y seraient applicables. Zéro ou plusieurs, donc multiplicité 0,N.

À la lumière de ces exemples nous pouvons conclure qu'il est plus simple en général d'établir les multiplicités de degré supérieur en Merise qu'en notation Chen ou UML. Nous constatons néanmoins par expérience que la logique de formulation des multiplicités des notations Chen ou UML mène à un nombre beaucoup moins grand d'associations de degré supérieur après application des règles de décomposition vues précédemment.

FIGURE 1-60 **Réservation auprès d'une société aérienne exprimée en Merise**



Nous nous limiterons à ces quelques considérations concernant les différences entre la notation de UML et la notation utilisée en Merise pour formuler un modèle entité-association. Nous pourrions élaborer d'avantage sur le plan comparatif en traitant notamment des contraintes inter-associations. L'objectif recherché étant d'offrir au lecteur un exposé des différences fondamentales, il ne nous semble pas à propos de pousser plus loin l'étude des

différences. Ces explications sur les différences fondamentales devraient permettre au lecteur déjà familier avec la notation UML d'interpréter fidèlement un modèle conceptuel exprimé selon la notation de Merise.

Notation UML

Selon l'organisme *Object Management Group* (OMG), maintenant responsable de documenter et de faire évoluer UML, ce dernier est « un langage visuel dédié à la spécification, la construction et la documentation des artefacts d'un système » [OMG 03]. D'ailleurs l'acronyme UML signifie *Unified Modeling Language*, soit « Langage unifié de modélisation ». C'est donc dire que UML intègre dans un même langage graphique des formalismes et des notations empruntés de méthodes de développement de système qui ont fait leur preuve dont notamment les méthodes dites *orientées objet*.

UML est suffisamment souple pour permettre qu'une notation serve à exprimer plusieurs types de modèles. Ainsi, en UML, la notation utilisée pour formuler un modèle conceptuel de données est appelée *diagramme de classes*. Un diagramme de classes, constitué essentiellement de cases rectangulaires, les classes, ainsi que d'arcs reliant les cases et affichant des multiplicités, peut exprimer :

- **Point de vue conceptuel** : le diagramme représente alors un *modèle entité-association* et est interprété comme décrivant les entités du monde réel ou du domaine à l'étude. Le diagramme, s'il s'inscrit dans le contexte de la conception d'une base de données, est généralement appelé un *modèle conceptuel de données*. En principe, chaque case du diagramme devrait porter la mention *Entity*. Cependant, en pratique, si le diagramme porte le nom modèle conceptuel de données, il n'est pas requis d'inscrire la mention *Entity* dans chaque case. Si le modèle vise par ailleurs la réalisation d'un logiciel, il sera appelé un *modèle du domaine* [LAR 05] où chaque case représente alors une classe conceptuelle du domaine.
- **Spécifications logicielles** : le diagramme représente un *modèle objet*. C'est-à-dire qu'il illustre des composants logiciels avec des spécifications et des interfaces qui pourront être programmées dans tout *langage orienté objet* comme par exemple C# ou Java. Chaque case représente une classe d'objets telle que définie dans un langage orienté objet.
- **Implémentation logicielle** : le diagramme décrit un *modèle d'implantation* du logiciel dans un langage orienté objet en spécifiant le comportement du logiciel grâce à des techniques spécifiques au langage.

Une autre notation que le diagramme de classes sera exploitée au cours des prochains chapitres. Il s'agit du *diagramme de cas d'utilisation* d'UML. Nous en ferons usage dans le contexte de l'étude d'une démarche systématique d'analyse, de conception et de réalisation d'une base de données.

UML est le fruit de travaux menés par plusieurs collaborateurs qui ont décidé d'en faire un standard ouvert, non propriétaire. Un premier groupe de travail de l'OMG a réalisé la mouture initiale de ce qui est devenu en 1997 un standard de facto, largement endossé par l'industrie de l'édition du logiciel : UML 1.0. L'année 2004 a vu l'émergence d'une nouvelle version d'UML, UML 2.0.

Cet ouvrage n'a pas la prétention de couvrir toutes les facettes de la notation UML. Bien au contraire. Nous devrons nous limiter aux éléments de la notation qui sont pertinents à la conception d'une base de données soit le diagramme de classes illustrant un modèle conceptuel de données et le diagramme de cas d'utilisation facilitant la définition des besoins en matière de données vitales. Le lecteur qui souhaite approfondir le sujet, notamment en ce qui à trait à la conception de logiciels avec UML, est invité à consulter l'ouvrage *UML 2.0* de Martin Fowler [FOW 03] paru en français aux éditions CampusPress.

LA PROCHAINE ÉTAPE

La modélisation conceptuelle des données n'est pas une fin en soi. Il s'agit d'une première étape dans la conception et la réalisation d'une base de données. Le modèle qui résulte de cette première étape doit être considéré comme un modèle stable qui décrit fidèlement le domaine ou le problème à l'étude. Cela signifie qu'il est notamment indépendant de la technologie utilisée pour réaliser la base de données recherchée.

L'élaboration d'un modèle conceptuel de données comporte de nombreux avantages. Il permet d'analyser les besoins en matière de données sans égard au choix d'une technologie de réalisation. Ce choix pourra être fait ultérieurement. Cela signifie qu'une fois le modèle conceptuel élaboré, le concepteur pourra faire le choix du type de SGBD sans remettre en question ce modèle. Mieux encore, cela permet à une organisation de changer de technologie sans remettre en question son modèle de données.

Comme nous le mentionnions en introduction, les SGBD appartiennent essentiellement à quatre grandes familles : hiérarchique, réseau, relationnel ou objet. Une organisation a le loisir de faire le choix d'un SGBD appartenant

à l'une de ces familles et dans cette famille d'un fournisseur en particulier, sans remettre en cause son modèle conceptuel de données. Néanmoins, le choix du SGBD aura un impact majeur sur la marche à suivre pour le mettre en œuvre. Le modèle conceptuel devra en effet être traduit dans un modèle de niveau plus spécifique, appelé le *modèle logique de données*, pour répondre aux exigences particulières de cette technologie.

Le chapitre suivant traite du passage du modèle conceptuel de données à un modèle logique de données qui tient compte de la technologie de réalisation qui sera utilisée : SGBD hiérarchique, réseau, relationnel ou objet. Puisque la technologie des SGBD relationnels domine actuellement les applications de gestion de données et ce pour encore longtemps, nous consacrerons le chapitre 2 à la démarche de production d'un modèle logique de données pour un SGBD relationnel à partir du modèle conceptuel de données.

Si nous devions être exhaustifs, il faudrait consacrer à chaque famille de SGBD un chapitre sur ce thème. Notre but n'étant pas l'exhaustivité, nous avons fait le choix des SGBD relationnels par souci de pragmatisme. Le chapitre 2 ne traite en conséquence que du passage du modèle conceptuel de données au modèle logique de données adapté au SGBD relationnel. Ce type de modèle logique de données est souvent appelé *modèle relationnel de données*.

AVANT DE FRANCHIR CETTE ÉTAPE : ASSURER LA VALIDATION DU MCD

L'étude du passage du modèle conceptuel de données au modèle logique de type relationnel est basée sur la prémisse que le modèle conceptuel à traduire est *valide*. Un MCD est valide s'il respecte au minimum les règles 1.1 à 1.4 introduites au cours du présent chapitre. Ces règles assurent la cohérence de la base de données dès le niveau conceptuel. La règle 1.5 qui traite de décomposition d'une association de degré supérieur pourrait être appliquée systématiquement mais son application n'est pas obligatoire pour produire un modèle relationnel correct. Plusieurs de ces règles sont basées sur les propriétés de dépendances fonctionnelles dont le modélisateur doit avoir une bonne maîtrise et sur lesquelles nous aurons le loisir de revenir au chapitre 2.

Les cinq règles peuvent servir de règles de validation d'un MCD et nous concluons ce chapitre en proposant une démarche de validation du modèle en cinq points :

1. Chaque entité doit posséder un identifiant (règle 1.1 d'identité). Cet identifiant est généralement explicite. Il peut être implicite dans trois cas : soit a) pour une entité d'association où il est formé par défaut de la combinaison des identifiants des entités de l'association, soit b) pour une entité de type composant où il est formé de la combinaison de l'identifiant du composant avec celui de son composite, soit c) pour une association d'héritage où une entité qui hérite des attributs d'une autre reçoit par conséquent l'identifiant de cette dernière et ne possède jamais d'identifiant qui lui est propre. Il faut éviter de faire apparaître explicitement un identifiant qui soit en fait implicite. Cela conduit à la transgression des règles 1.3 et 1.4.
2. Chaque attribut ne peut avoir qu'une valeur à la fois et cette donnée doit avoir un caractère élémentaire, c'est-à-dire posséder un type de données simple (règle 1.2 de description).
3. Un attribut ne peut figurer qu'une seule fois dans le diagramme entité-association (règle 1.3 de non-redondance). Prendre garde aux *synonymes*, la même donnée sous deux noms différents, et à la *polysémie*, une même appellation pour deux données différentes.
4. Les attributs d'une entité décrivent bien cette entité et lui appartiennent en propre. Aucun ne peut appartenir à une autre entité (règle 1.4 de construction). La valeur de chaque attribut est déterminée de manière unique par la valeur de l'identifiant. Cette règle possède un corollaire : il faut éviter de répliquer un attribut dans une entité provenant d'une deuxième entité pour signifier qu'elles sont liées. Pour ce faire, il faut employer une association.
5. Il est souhaitable de décomposer les associations de degré supérieur le cas échéant (règle 1.5 de décomposition). Deux situations se prêtent à la décomposition : a) une des multiplicités maximales est égale à 1 ; b) il existe une dépendance fonctionnelle entre deux identifiants parmi les entités associées.

EXERCICES DE MODÉLISATION CONCEPTUELLE DES DONNÉES

EXERCICE 1-1

Cet exercice consiste à modéliser les données requises pour élaborer un contrat de location dans une entreprise de location de véhicules automobiles. La structure de données ci-après est extraite du document *Contrat de location*. La structure est fort simple puisqu'elle ne comporte ni sous-structures, ni structures répétitives.

Bien que le contrat ne concerne qu'un seul véhicule et un seul client, le même véhicule peut faire l'objet de plusieurs locations au même client ou à des clients différents. Toute la tarification, que ce soit pour la location ou les assurances, est établie pour chaque véhicule et non pour le modèle auquel il appartient. Cela signifie que la tarification est basée sur les caractéristiques propres au véhicule : nombre de portes, type de transmission, etc. On prend pour acquis que le client n'est titulaire que d'un seul permis de conduire.

Faire le modèle conceptuel des données relatives à ce domaine d'application à partir des données du *Contrat de location*. Il s'agit d'un exercice relativement simple car on ne devrait y retrouver que des associations binaires, sans entités d'association. Le nombre de données à considérer est néanmoins important.

Suggestions : regrouper dans une même entité toutes les données relatives à la tarification applicable à un véhicule et donner un identifiant artificiel à cette entité, par exemple **No de tarif** ; faire usage d'entités de description pour les données de type énuméré : marque, modèle, catégorie.

```
Descriptif du document: Contrat de location
- No immatriculation
- No série
- Marque                    (Honda, Chevrolet, …)
- Modèle                    (CX, CRX, …)
- Année
- Climatisation?
- Automatique?
- Nombre portes
- Kilométrage actuel
- Catégorie                 (Économique, compacte, …)
- Tarif Horaire
- Tarif quotidien
- Tarif hebdomadaire
- Tarif kilométrage
- Tarif assurance collision quotidien
- Montant franchise collision
- Tarif suppression de franchise quotidien
- Nom du client
- Adresse client
- Téléphone client
- No permis de conduire
- Province permis
- Pays permis
- No contrat location
```

```
- Date contrat
- Lieu prise possession
- Heure prise possession
- Date remise
- Lieu remise
- Heure remise
- Code de rabais appliqué
- Kilométrage inclus
- Assurance collision?
- Suppression franchise?
- Nombre de jours location
- Heures en sus
- Kilomètres parcourus
- Montant facturé
```

EXERCICE 1-2

La structure de données suivante est extraite d'un document comportant des données sur le dépôt d'une candidature à un poste dans une organisation. Comme on peut le noter facilement à la consultation de la structure de données, un candidat peut pratiquer plusieurs langues, avoir de multiples centres d'intérêt, posséder plusieurs diplômes, avoir eu plusieurs employeurs chez qui il a pu exercer plusieurs fonctions.

Descriptif du document: ***Dépôt de candidature***

- No affichage du poste
- Nom du poste
- No du candidat
- Nom candidat
- Prénom candidat
- No téléphone
- Date de naissance
- **Langue pratiquée (*)**
  - Langue
  - Niveau de connaissance (parlé, écrit, les deux)
- **Centre d'intérêt (*)**
  - Désignation (sport, musique,…)
- **Employeur (*)**
  - Raison sociale entreprise
  - Date d'embauche entreprise
  - Date de départ de l'entreprise
  - **Fonction exercée (*)**
    - Désignation fonction
    - Date d'entrée en fonction
    - Date de départ de la fonction
    - Salaire à la fin
- **Diplôme (*)**
  - Désignation diplôme
  - Date d'obtention
  - Nom institution

Le candidat peut avoir été réembauché plusieurs fois chez un même employeur. Le candidat peut avoir exercé la même fonction chez plusieurs employeurs et avoir exercé la même fonction plus d'une fois chez un même employeur. Il ne peut cependant pas avoir obtenu le même diplôme plus d'une fois.

Il s'agit d'élaborer un modèle conceptuel de données qui reflète ces exigences. Il peut comporter une association de degré 3 qui peut être réduite à deux associations binaires.

EXERCICE 1-3

Cet exercice consiste à modéliser les données relatives à l'organisation de courses hippiques au cours d'une saison. Le descriptif du document qui suit fait état des données conservées pour chaque course d'une saison, sans égard au champ de course où elle se déroule.

```
Descriptif du document: Course de chevaux
- No course
- Saison
- Désignation de la course
- Type de course            (tiercé, quarté, …)
- Catégorie de course       (trot attelé, trot monté, obstacle)
- Date de la course
- Dotation de la course en dollars
- Nom du champ de course
- Cheval participant (*)
-      Nom du cheval
-      Sexe du cheval
-      Date de naissance du cheval
-      Gains du cheval cette saison
-      Nom du jockey
-      No licence du jockey
-      No de dossard pour la course
-      Propriétaire (*)
-             Nom propriétaire du cheval
```

Le descriptif met en évidence qu'une course ne se déroule que dans un seul lieu (le champ de course), qu'elle est d'un seul type et d'une seule catégorie, qu'elle implique plusieurs chevaux participants, qu'un cheval ne peut être conduit que par un seul jockey dans la course mais peut être la propriété de plusieurs personnes.

Un champ de course peut recevoir tous les types de courses et être aménagé pour toutes les catégories de course. Le numéro de dossard est porté par le jockey mais identifie à la fois le cheval et le jockey dans une course.

Il est considéré vital de savoir qu'un cheval est le parent d'un autre cheval pour établir la lignée d'un animal.

Cette exigence devrait donner lieu à une association réflexive dans le modèle conceptuel. Par ailleurs le modèle peut comporter une association de degré 3 comportant une entité d'association. Elle peut être réduite à deux associations binaires.

EXERCICE 1-4

Considérons la structure de données suivante faisant état de données relatives à un ordre de production dans une usine.

```
Descriptif du document: Ordre de production
- No ordre production
- Statut de l'ordre          (Ferme, Lancé, Terminé)
- No produit
- Quantité à produire
- Quantité produite
- No processus
- Type de processus          (Montage ou Usinage)
- Responsable processus
- Date dernier changement
- Opération du processus (*)
-       No de séquence       (1, 2, 3, etc.)
-       Description
-       Temps standard
-       Date planifiée
-       Date de fin prévue
-       Produit utilisé (*)
-              No produit
-              Quantité utilisée
-       Machine utilisée (*)
-              No machine
-              Description
-              Coût utilisation/min.
-              Date prévue utilisation
-              Durée d'utilisation
-              Quantité réalisée
-              Quantité rebutée
-              Outillage utilisé (*)
-                     No outillage
-                     Description   (Ex. mèche 10 mm)
-                     Quantité utilisée
```

Chaque ordre de production concerne un seul produit qui doit être monté ou usiné selon la **Quantité à produire**. Un seul processus est planifié pour réaliser un produit. Le processus comporte une séquence d'opérations, en quelque sorte les étapes du processus de production, et chaque opération est planifiée avec une **Date planifiée** et une **Date de fin prévue**. Pour chaque opération, un autre produit peut être utilisé. C'est le cas lorsque l'ordre concerne le montage d'un produit. Le cas échéant, dans une opération, la quantité de chaque produit utilisée est spécifiée (**Produit- Quantité utilisée**)

La notion de produit est large. Un produit peut être en effet soit monté ou usiné sur place soit acheté auprès d'un fournisseur.

Une opération n'appartient qu'à un seul processus et ne peut exister que dans le contexte de ce processus (Durée de vie identique). Chaque opération dans un processus peut faire appel à des machines de l'usine. Dans le cadre de la planification d'un ordre de production, on doit préciser quelles machines seront utilisées et pour chaque machine, la date et la durée d'utilisation (**Date prévue utilisation**, **Durée d'utilisation**). Pour chaque

machine qui sera utilisée à l'intérieur d'une opération planifiée dans l'ordre, on souhaite conserver des données sur la quantité de pièces produites (**Quantité réalisée**) et la quantité de pièces rejetées (**Quantité rebutée**). De plus on souhaite savoir, à la suite de la réalisation d'une opération, quels sont les outillages utilisés sur chaque machine et en quelle quantité (**Outillage- Quantité utilisée**). Par exemple, on pourrait avoir utilisé une mèche 10 mm et deux mèches 12 mm sur la perceuse numéro 20.

Il s'agit d'élaborer le modèle conceptuel des données du document Ordre de production. On devrait y retrouver notamment une association de composition et une association ternaire.

EXERCICE 1-5

Pour faire suite à l'exercice précédent, il faut modéliser le concept de *produit* considérant qu'un produit peut être composé d'autres produits et qu'un produit présent dans une composition peut avoir un ou des substituts. Cette situation exige une association binaire réflexive sur l'entité **Produit**. On ne peut faire appel ici à une association de composition réflexive car un produit peut exister indépendamment d'un autre produit. On traite dans cet exercice de catégorie de produits. Un produit a comme identifiant **No produit** et il reflète ainsi la catégorie regroupant tous les exemplaires d'un produit en particulier, par exemple un lave-vaisselle inscrit au catalogue sous le no de produit H8976, et non pas un exemplaire en particulier comme le lave-vaisselle de type H8976 portant le no de série 78876254342 produit le 6 juin 2006.

Tout produit peut être le composant d'un autre produit ou peut être un substitut dans la composition d'un autre.

La structure de données suivante explicite la nomenclature d'un produit, soit sa structure et ses éléments de substitution.

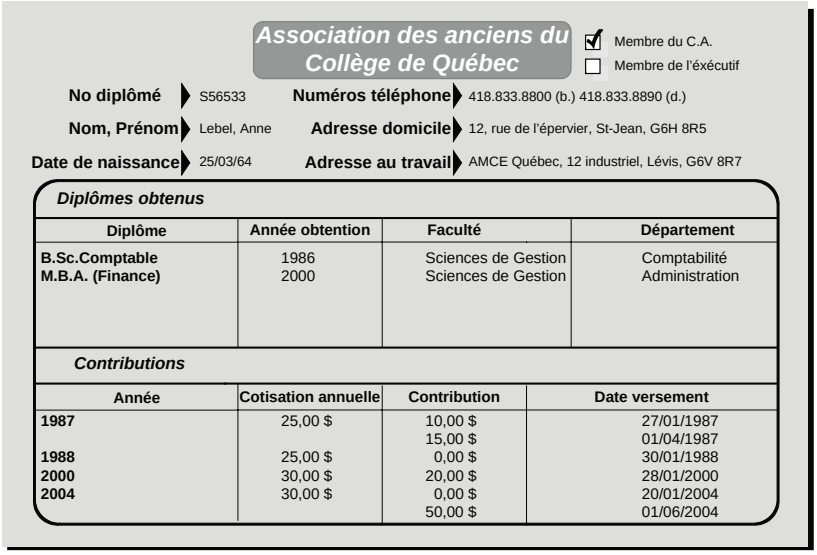
```
Descriptif du document: Nomenclature de produit
- No Produit
- Désignation du produit
- Composition (*)
-       No produit
-       Quantité du produit
-       Substitut du produit (*)
-               No produit
```

EXERCICE 1-6

Cet exercice consiste à établir le descriptif des deux documents faisant l'objet du cas Collège de Québec présenté ci-après. En fonction des exigences du cas, veuillez élaborer le modèle conceptuel des données.

Le Collège de Québec s'est doté d'une association de diplômés dont la mission première est de faire des levées de fonds pour les activités du Collège. Tous les anciens du Collège sont sollicités annuellement par la poste pour verser une cotisation leur permettant d'être membre de l'association. Les diplômés ont le loisir de verser une contribution volontaire en plus de leur cotisation annuelle. Les contributions et cotisations peuvent être payées en plusieurs versements.

Pour faire le suivi des adhésions et comptabiliser les contributions de ses membres, l'association dispose d'une fiche sur laquelle sont inscrites les données relatives au membre.



Association des anciens du Collège de Québec

☑ Membre du C.A.
☐ Membre de l'éxécutif

No diplômé S56533 — **Numéros téléphone** 418.833.8800 (b.) 418.833.8890 (d.)

Nom, Prénom Lebel, Anne — **Adresse domicile** 12, rue de l'épervier, St-Jean, G6H 8R5

Date de naissance 25/03/64 — **Adresse au travail** AMCE Québec, 12 industriel, Lévis, G6V 8R7

Diplômes obtenus

Diplôme	Année obtention	Faculté	Département
B.Sc.Comptable	1986	Sciences de Gestion	Comptabilité
M.B.A. (Finance)	2000	Sciences de Gestion	Administration

Contributions

Année	Cotisation annuelle	Contribution	Date versement
1987	25,00 $	10,00 $	27/01/1987
		15,00 $	01/04/1987
1988	25,00 $	0,00 $	30/01/1988
2000	30,00 $	20,00 $	28/01/2000
2004	30,00 $	0,00 $	20/01/2004
		50,00 $	01/06/2004

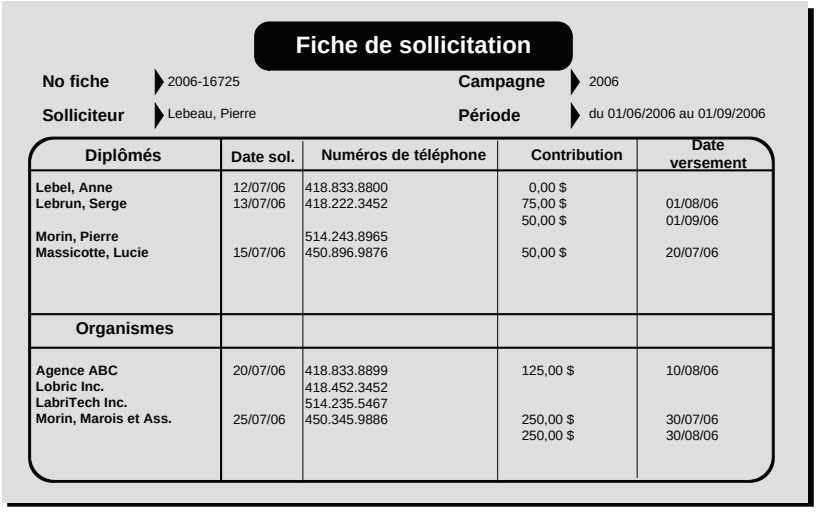
La fiche du membre permet de savoir notamment si le membre siège au conseil d'administration ou au comité exécutif de l'association. Chaque diplôme obtenu au Collège est inscrit sur la fiche. De plus, on y fait état des contributions annuelles du membre, sous forme de cotisations ou non. Les contributions peuvent se faire en plusieurs versements et les dates de chaque versement sont conservées sur la fiche.

L'association accorde une grande importance à l'enregistrement de chaque versement, que le versement soit relatif à une simple cotisation annuelle, à une contribution sans cotisation ou à une cotisation avec contribution.

Le montant de la cotisation annuelle du membre dépend du type de diplôme obtenu. Si le diplômé s'est vu décerner plusieurs diplômes, c'est le diplôme qui commande la plus haute cotisation qui établit sa cotisation annuelle.

Chaque année, l'association lance une campagne de levée de fonds et fait appel à ses membres pour solliciter les diplômés de l'association ainsi que des organismes du milieu susceptibles de contribuer financièrement à ses activités. Plusieurs périodes de levée de fonds peuvent être programmées au cours d'une année. Pour faciliter le travail des solliciteurs une **Fiche de sollicitation** est complétée pour chaque période de levée de fonds. La fiche est préparée à l'intention d'un solliciteur qui est forcément membre de l'association. On y fait état des diplômés et des organismes du milieu que le membre solliciteur doit contacter ainsi que des numéros de téléphone de ces derniers.

Lorsque le solliciteur contacte un diplômé ou un représentant d'organisme apparaissant sur la fiche qu'on lui a remise, il complète celle-ci en y inscrivant la date à laquelle la sollicitation a été faite et, le cas échéant, la liste des versements convenus et la date de chacun de ces versements.



Fiche de sollicitation

No fiche 2006-16725 **Campagne** 2006

Solliciteur Lebeau, Pierre **Période** du 01/06/2006 au 01/09/2006

Diplômés	Date sol.	Numéros de téléphone	Contribution	Date versement
Lebel, Anne	12/07/06	418.833.8800	0,00 $	
Lebrun, Serge	13/07/06	418.222.3452	75,00 $	01/08/06
			50,00 $	01/09/06
Morin, Pierre		514.243.8965		
Massicotte, Lucie	15/07/06	450.896.9876	50,00 $	20/07/06
Organismes				
Agence ABC	20/07/06	418.833.8899	125,00 $	10/08/06
Lobric Inc.		418.452.3452		
LabriTech Inc.		514.235.5467		
Morin, Marois et Ass.	25/07/06	450.345.9886	250,00 $	30/07/06
			250,00 $	30/08/06

Comme l'illustre un exemplaire de la fiche montré ci haut, une sollicitation peut ne pas donner lieu à une contribution. Elle peut par ailleurs donner lieu à un ou plusieurs versements.

Chaque fiche à une identification unique et s'applique à une période de levée de fonds (date de début, date de fin) s'inscrivant à l'intérieur d'une campagne annuelle.

Il s'agit de rédiger le *Descriptif du document* des deux types de fiches compilées par l'association et élaborer ensuite un modèle conceptuel des données vitales au fonctionnement de celle-ci.

Suggestions : Étant donnée l'importance que revêt chaque versement, que ce soit pour le paiement d'une cotisation ou non, nous suggérons de créer dans le descriptif du document **Association des anciens** une structure répétitive regroupant les données suivantes :

```
- ...
- Versement (*)
-       Année
-       Cotisation?
-       Date versement
-       Montant versement
```

Cette structure représente les versements effectués par le membre d'une année à l'autre et permet de savoir si le versement est fait pour acquitter la cotisation annuelle ou non, selon la valeur du booléen **Cotisation ?**.

Quant au descriptif de la **Fiche de sollicitation**, on note que les mêmes données y sont compilées qu'il s'agisse d'un membre ou d'un organisme. En conséquence, nous suggérons que la structure répétitive suivante soit intégrée au descriptif de manière à établir, pour chaque sollicitation, à l'aide de la donnée booléenne **Diplômé ?**, s'il s'agit d'un diplômé ou d'un organisme. **Téléphone** est considérée comme une structure répétitive car les membres ou les organismes sollicités peuvent posséder plus d'un numéro de téléphone.

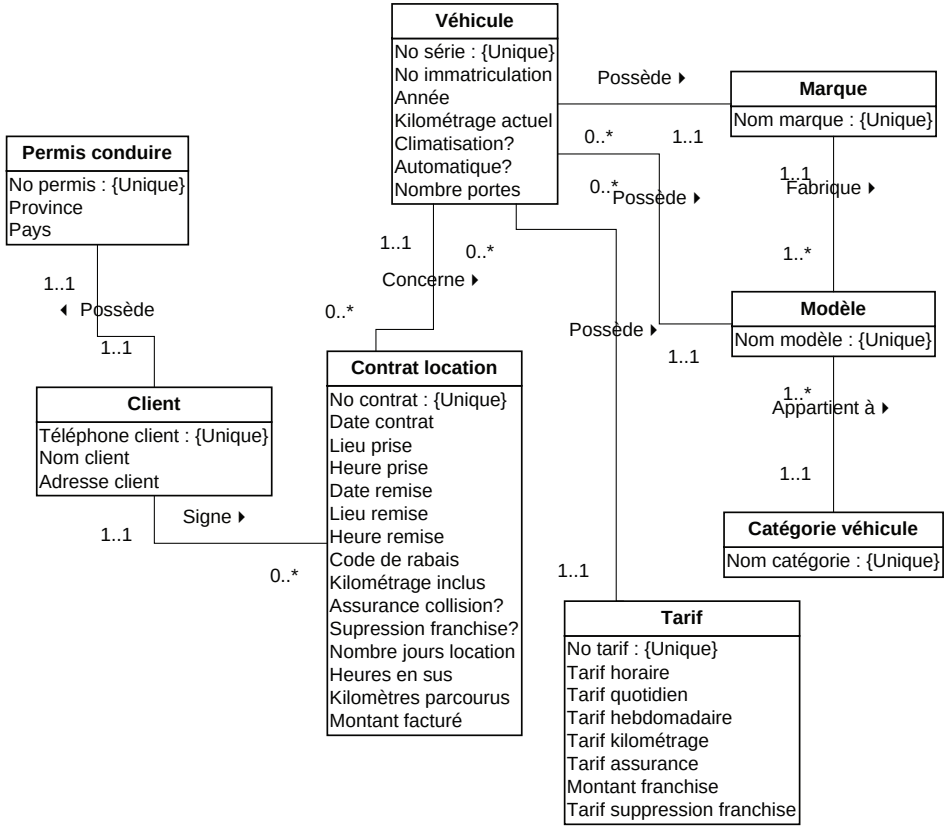
```
- ...
- Sollicitation (*)
-       Nom sollicité
-       Diplômé?
-       Téléphone (*)
-               No téléphone
- ...
```

Quant au modèle conceptuel, il s'agit de refléter la réalité des institutions d'enseignement : un diplôme est obtenu dans un seul Département rattaché à une Faculté. Une association de composition peut être envisagée pour mettre en évidence qu'une période de campagne est composée de sollicitations dépendant intégralement de leur période. Deux contraintes inter-associations peuvent être utilisées. Une première pour indiquer qu'un versement porte soit sur une cotisation soit sur une simple contribution et une deuxième permettant de montrer qu'une sollicitation concerne soit un organisme soit un diplômé.

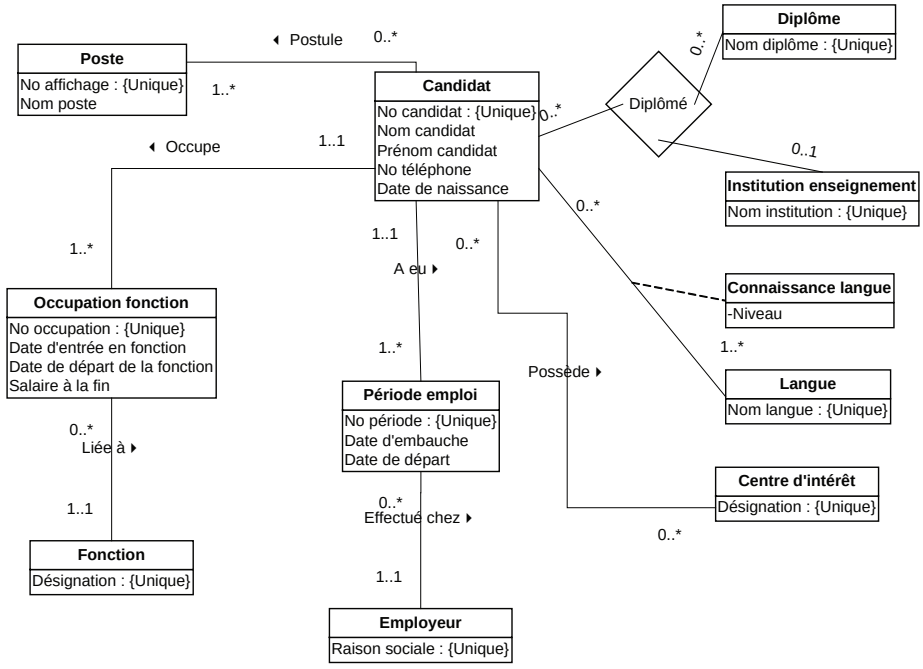
SOLUTIONS DES EXERCICES DE MODÉLISATION CONCEPTUELLE DES DONNÉES

EXERCICE 1-1

Tel que suggéré, l'entité **Tarif** est présente dans le modèle et elle est associée à l'entité **Véhicule**. Les entités de description **Marque**, **Modèle**, **Catégorie** sont présentes pour décrire soit le véhicule, soit le modèle (**Catégorie**). **Permis de conduire** est traitée comme entité distincte de **Client** car le permis a son existence propre. Une association appelée **Fabrique** entre **Marque** et **Modèle** est requise pour savoir à quelle marque appartient un modèle, car il est impossible de l'établir transitivement via l'entité **Véhicule** puisque qu'un modèle est lié à plusieurs véhicules. L'entité **Contrat de location** ne comporte que les données spécifiques à un contrat de location. Les autres éléments du contrat sont donnés par des associations binaires menant à une seule occurrence de **Client**, une seule occurrence de **Véhicule**, une seule occurrence de **Permis**, une seule occurrence de **Tarif** (transitivement via **Véhicule**).



EXERCICE 1-2 Une première version de la solution est donnée ci-après.

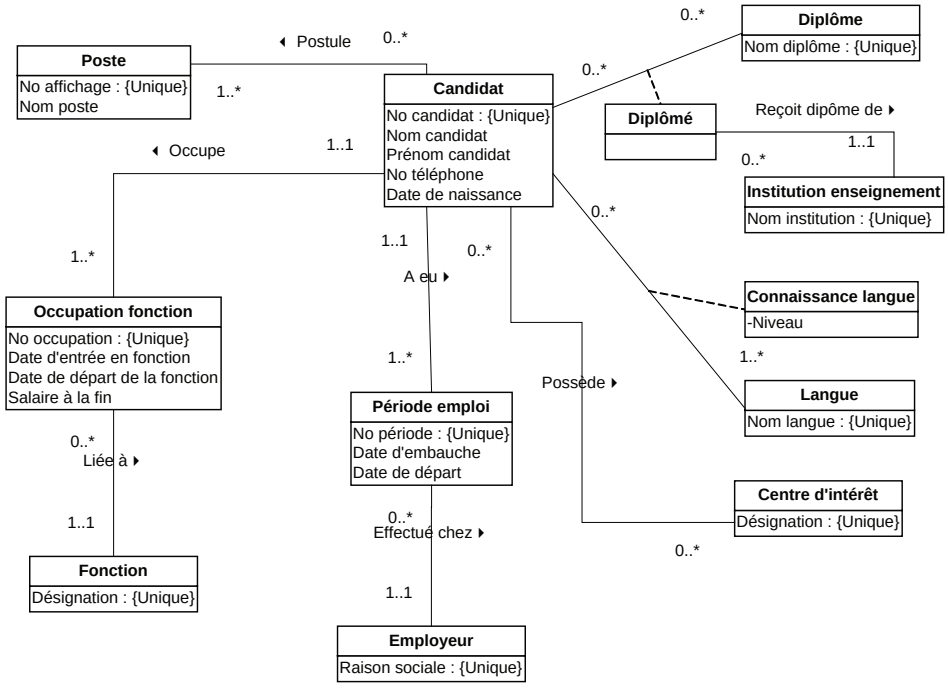


Poste, Candidat, Diplôme, Institution d'enseignement, Langue, Centre d'intérêt, Employeur et **Fonction** sont tous des objets concrets ou abstraits qui ont leur existence propre et qui sont des sujets d'intérêt dans ce domaine.

Occupation fonction aurait pu être considérée comme entité d'association (entité faible) si un candidat ne peut occuper la même fonction plus d'une fois. Mais comme ce n'est pas le cas, **Occupation fonction** doit être considérée comme une entité forte, avec son propre identifiant que nous devons créer artificiellement (**No occupation**). Il en va de même pour **Période emploi** qui ne peut être vue comme entité faible : un candidat et un employeur donné déterminent plusieurs périodes d'emploi. Il n'y a donc aucune dépendance fonctionnelle entre une période et la combinaison **No candidat-Raison sociale**.

On note la présence d'une association ternaire, **Diplômé**, associant un candidat, le diplôme et l'institution d'obtention. Cette association à une multiplicité maximale de 1 côté **Institution** puisque le candidat ne peut détenir plus d'une fois le même diplôme et qu'il ne peut provenir que d'une seule institution.

Dans un tel contexte il est possible de réduire le degré de l'association de 3 à 2 en faisant de **Diplômé** une entité d'association sans attribut, liée directement à **Institution d'enseignement**.



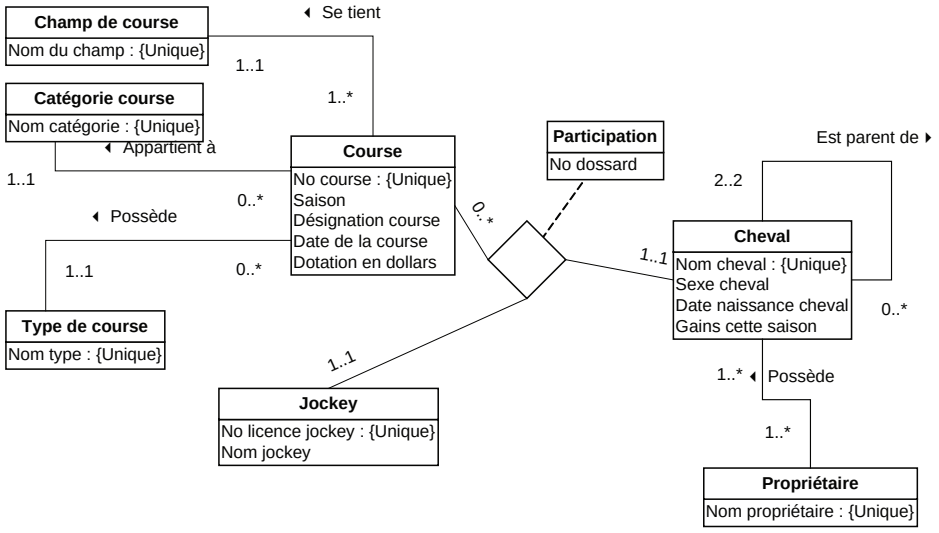
Les deux modèles ont la même valeur sur le plan sémantique. Il incombe au modélisateur de décider lequel reflète le plus clairement et le plus simplement la réalité du domaine étudié.

EXERCICE 1-3

Une première version du modèle conceptuel est proposée ci-après. Il comporte une association de degré 3. Comme nous le verrons plus bas, cette association peut facilement se réduire à deux associations binaires compte tenu de la présence de multiplicités maximales valant 1 sur deux arcs de l'association.

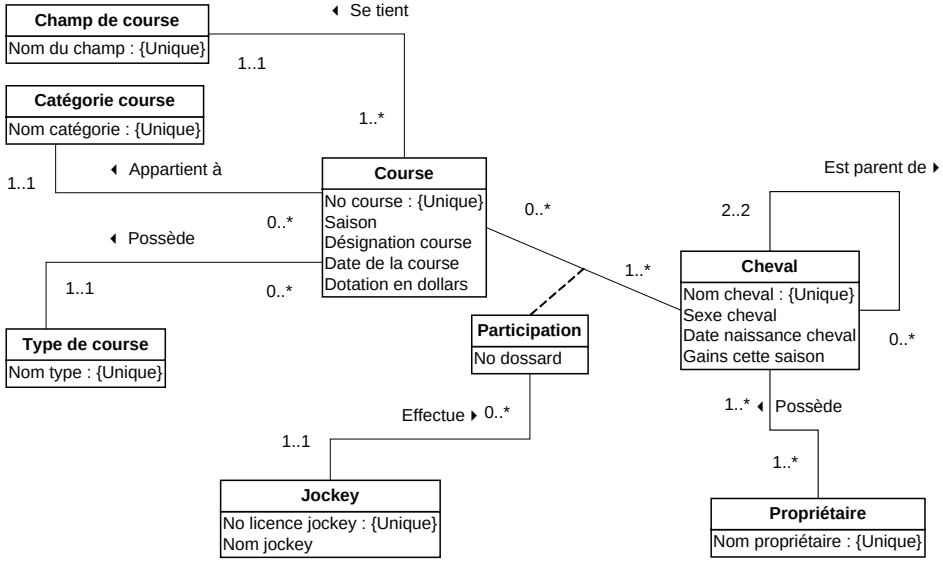
Champ de course, Course, Jockey, Cheval et **Propriétaire** sont toutes des entités pertinentes à ce domaine. Elles représentent des objets concrets ou abstraits ayant leur existence propre, donc qui ont une existence indépendante les uns des autres. Les entités **Catégorie de course** et **Type de course** sont tout aussi pertinentes mais pour une autre raison. Elles représentent des entités dites de *description*. Leur rôle consiste à décrire ou à qualifier d'autres entités par association. Ce genre d'entité permet notamment de créer une nomenclature pour classifier ou organiser d'autres entités.

L'entité **Participation** est une entité faible dont une occurrence ne peut exister que s'il y a conjugaison d'un cheval, d'un jockey et d'une course. Le numéro du dossard ne peu exister que si ces trois occurrences sont connues. L'association de degré 3 montre qu'un cheval dans une course ne peut être conduit que par un seul jockey. De plus, un jockey qui participe à une course ne peut conduire qu'un seul cheval. On peut dès lors



envisager de réduire cette association ternaire en associant l'entité d'association **Participation** soit à **Jockey**, soit à **Cheval**. Nous choisissons arbitrairement de le faire avec **Jockey** (voir modèle suivant).

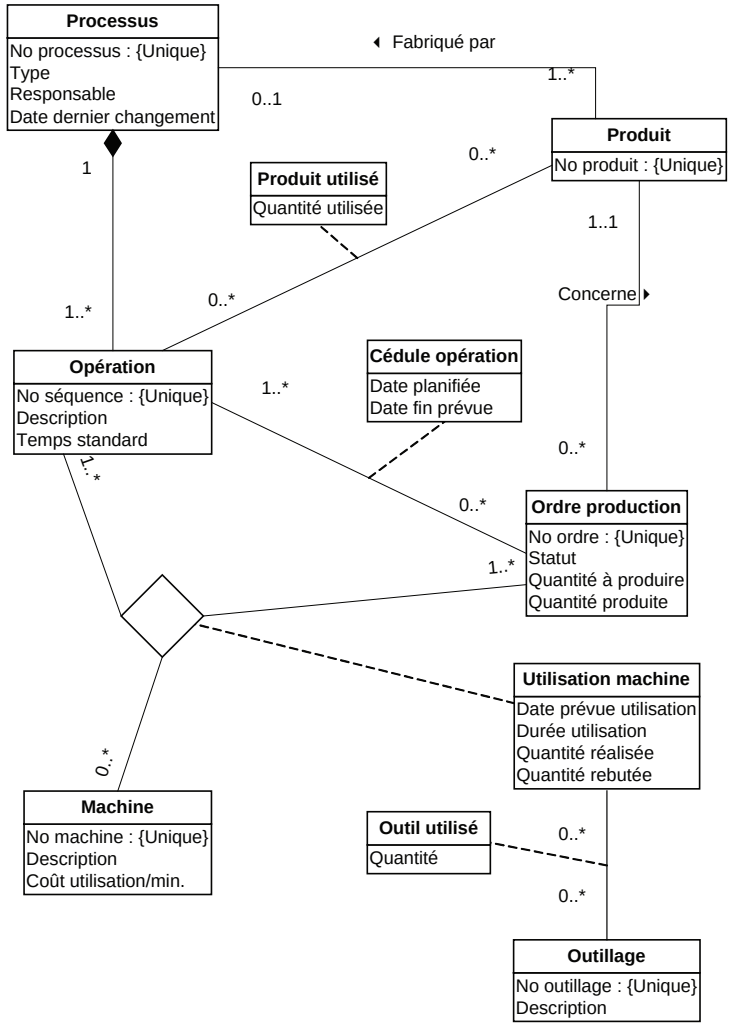
Enfin, une association réflexive sur l'entité **Cheval** établit le lien de filiation exigé entre les chevaux. On note la multiplicité 2..2 conforme à la réalité de la procréation.



EXERCICE 1-4

Ordre de production, Produit, Processus, Opération, Machine et **Outillage** sont des objets d'intérêt dans ce cas. Ils ont tous une existence bien réelle et possèdent un identifiant. Ce sont sans aucun doute des entités fortes. **Opération** est un composant de **Processus**. Son identifiant réel est la combinaison **No Processus-No séquence**.

Un ordre de production concerne un et un seul produit mais en contrepartie, un produit pourrait ne pas faire l'objet d'un ordre car certains produits sont acquis à partir de fournisseurs. Un produit est fabriqué par au plus un processus mais rien n'indique dans ce cas qu'un processus peut permettre de réaliser un seul produit ou au contraire plus d'un produit. Nous avons retenu cette dernière hypothèse qui est moins contraignante.



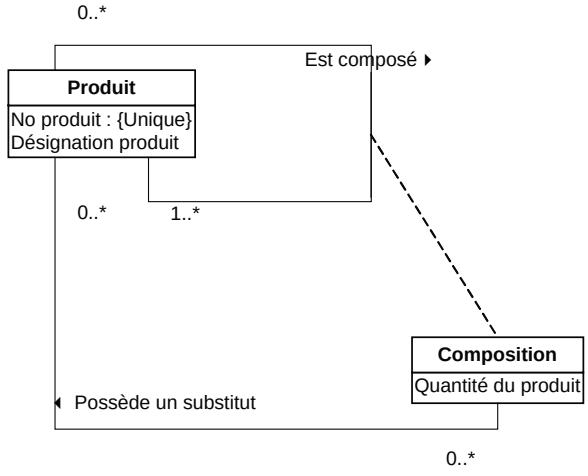
La quantité d'un produit utilisé dans une opération dépend à la fois de l'opération et du produit (entité faible), d'où la présence d'une entité d'association. Il en va de même de **Date planifiée** et **Date de fin prévue** qui dépendent à la fois de **Opération** et **Ordre de production**. Ces deux dernières données sont rattachées à une entité d'association.

L'association ternaire **Utilisation machine** s'impose car les quatre données liées à cette association dépendent toutes fonctionnellement des trois entités à la fois. De plus, les multiplicités maximales sur les trois arcs sont obligatoirement plusieurs (*). En effet, si un ordre de production est lié à une opération, plusieurs machines peuvent être en cause. Si un ordre de production est lié à une machine, ce peut être pour plusieurs opérations. Enfin si une machine est liée à une opération, ce pourrait être dans le cadre de plusieurs ordres de production.

L'entité d'association **Utilisation machine** doit être associée à l'entité **Outillage** pour établir, à chaque utilisation de machine, la quantité d'outillage utilisé. Il s'agit bien d'une association plusieurs à plusieurs car un type d'outillage peut faire l'objet de plusieurs utilisations et une utilisation de machine donne lieu à la consommation de plusieurs outillages différents.

EXERCICE 1-5

Le modèle suivant reflète assez fidèlement les exigences du problème. L'entité **Produit** est associée à elle-même pour refléter la composition du produit et l'entité d'association **Composition** retient la quantité d'un produit donné à titre de composant d'un autre. Un élément de composition peut avoir un substitut, ce qui est exprimé par une association binaire entre **Composition** et **Produit**.



Comme par définition une association binaire réflexive relie deux occurrences de la même entité, le modèle n'interdit point qu'un produit soit composé de lui-même, ce qui ne semble pas être le cas en toute logique même si l'exposé de la situation n'est pas explicite à cet égard.

Le modèle ne permet pas d'exclure qu'un substitut dans un produit soit le produit substitué lui-même. De plus, même s'il semble illogique que si **A** est composé de **B**, **B** ait comme substitut **A** dans la composition de **A**, cette possibilité n'est pas interdite en vertu du modèle ci-dessus. Ce qui revient à dire qu'un produit pourrait être composé de lui-même.

Il est malheureusement impossible de faire appel à une contrainte inter-associations pour exclure la possibilité qu'un produit soit composé de lui-même et qu'un substitut dans un produit ne soit le produit substitué lui-même. Ces dernières contraintes ne portent en fait que sur une association à la fois.

Nous devons donc nous résoudre à adopter un modèle qui ne peut prendre en charge toutes les contraintes évoquées.

EXERCICE 1-6

Les descriptifs des documents reflètent les suggestions faites à la fin de l'exercice. Le premier montre que les versements faits par un diplômé sont de première importance et la donnée **Cotisation ?** permet d'établir qu'il s'agit d'un versement pour une cotisation annuelle ou non.

Descriptif du document: ***Association des anciens***

- No diplômé
- Nom diplômé
- Prénom diplômé
- Date de naissance
- No téléphone bureau
- No téléphone domicile
- Adresse domicile
- Adresse travail
- Membre CA?
- Membre exécutif?
- **Diplôme (*)**
- Désignation diplôme
- Année obtention
- Département
- Faculté
- **Versement (*)**
- Année
- Cotisation?
- Date versement
- Montant versement

Le deuxième descriptif, **Fiche de sollicitation**, fait ressortir l'importance de la notion de sollicitation. Il montre qu'une sollicitation donne lieu le cas échéant à des versements de la part de la personne sollicitée.

Puisque le solliciteur est forcément un diplômé, le nom et le prénom présents sur la fiche pour l'identifier portent forcément la même appellation que celle donnée dans le descriptif précédent. De plus le descriptif indique que la structure **Sollicitation** peut comporter le Nom et le Prénom d'un diplômé ***ou bien*** le Nom d'un organisme, pas les deux à la fois.

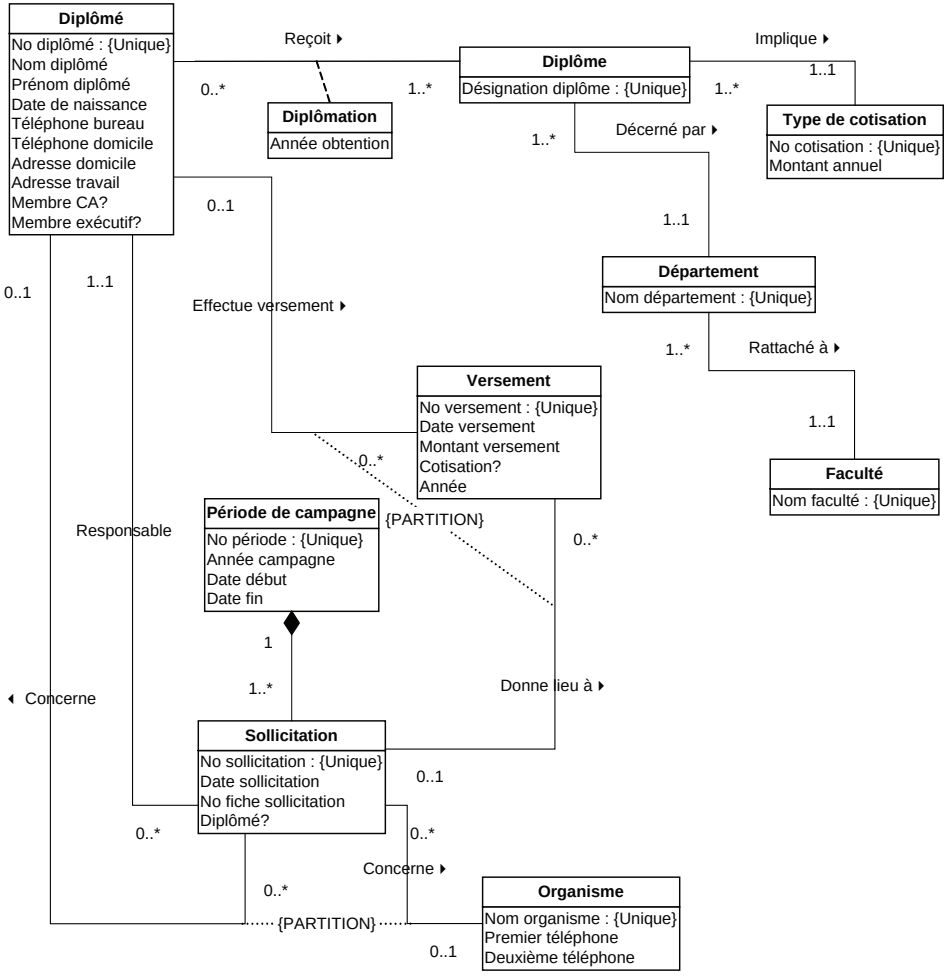
La donnée **Diplômé ?** sert à déterminer si la sollicitation s'adresse à un diplômé. Dans le cas contraire il s'agit d'un organisme.

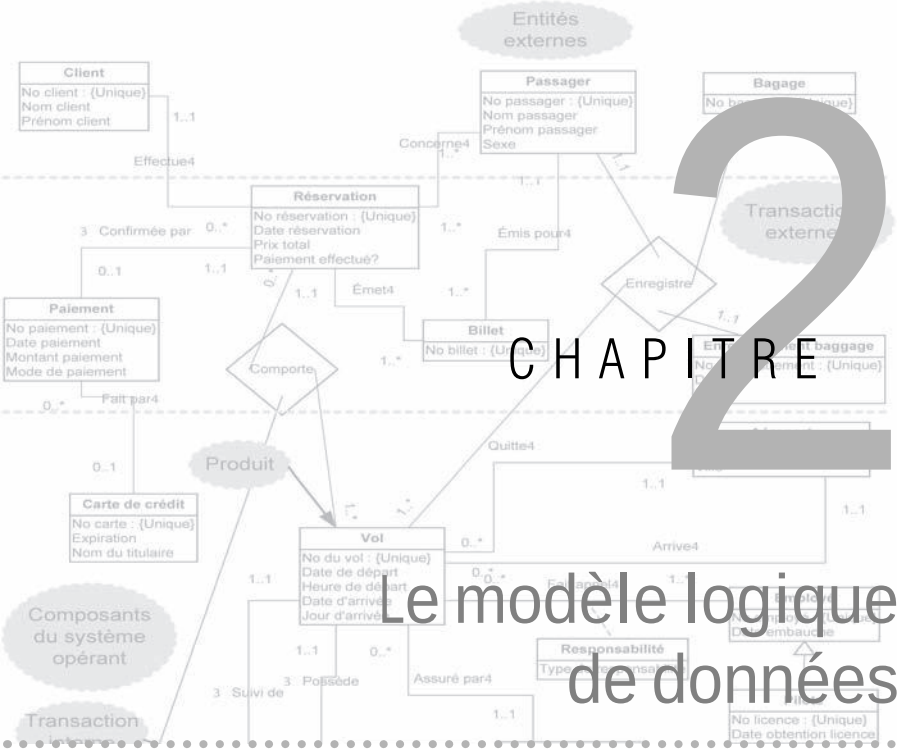
Descriptif du document: ***Fiche de sollicitation***
- No fiche
- Nom diplômé
- Prénom diplômé
- Année
- Début période
- Fin période
- **Sollicitation (*)**
- Nom, Prénom diplômé OU Nom organisme
- Diplômé?
- Date sollicitation
- **Téléphone (*)**
- No téléphone
- **Versement (*)**
- Date versement
- Montant versement

Diplômé, **Diplôme**, **Faculté**, **Département** et **Organisme** sont toutes des entités d'intérêt qui ont une existence propre, bien que certaines ne sont décrites que par un seul attribut. **Type de cotisation** est une entité de description permettant de connaître la cotisation annuelle à verser selon le diplôme obtenu. Cette donnée est vitale pour déterminer annuellement le solde de la cotisation à payer.

L'entité **Période de campagne** est traitée comme un composite constitué de l'entité **Sollicitation**, une occurrence de **Sollicitation** ne pouvant exister que dans le contexte d'une période de campagne. L'entité **Sollicitation** est doublement associée à **Diplômé**. D'une part pour déterminer quel diplômé est responsable de la sollicitation et d'autre part pour savoir le cas échéant quel est le diplômé sollicité. La sollicitation peut concerner soit un diplômé, soit un organisme, ce qui est spécifié par une contrainte inter-associations de PARTITION entre les deux associations **Concerne**. Une contrainte du même type est utilisée pour illustrer qu'un versement est effectué soit par suite d'une sollicitation directe dans le cadre d'une période de campagne de levée de fonds, soit lors de l'inscription annuelle effectuée par la poste.

L'entité **Versement** comporte les données requises pour établir le solde à payer d'une cotisation annuelle car on y retrouve les attributs **Cotisations ?** identifiant les versements portant sur une cotisation et **Année** qui détermine l'année de cotisation couverte par le paiement. Le calcul du solde se fait simplement en établissant d'abord le montant de cotisation le plus élevé parmi les diplômes reçus par un diplômé puis en faisant ensuite la somme des versements de cotisation effectués par le diplômé au cours d'une année.


Diplômé
No diplômé : {Unique}
Nom diplômé
Prénom diplômé
Date de naissance
Téléphone bureau
Téléphone domicile
Adresse domicile
Adresse travail
Membre CA?
Membre exécutif?
Reçoit
0..*
1..*
Diplômation
Année obtention
Diplôme
Désignation diplôme : {Unique}
Implique
1..1
1..*
Type de cotisation
No cotisation : {Unique}
Montant annuel
Décerné par
1..*
1..1
Département
Nom département : {Unique}
0..1
1..1
0..1
Effectue versement
1..*
Rattaché à
Versement
No versement : {Unique}
Date versement
Montant versement
Cotisation?
Année
1..1
Faculté
Nom faculté : {Unique}
0..*
{PARTITION}
Responsable
Période de campagne
No période : {Unique}
Année campagne
Date début
Date fin
0..*
1
1..*
Concerne
Donne lieu à
Sollicitation
No sollicitation : {Unique}
Date sollicitation
No fiche sollicitation
Diplômé?
0..1
0..*
0..*
Concerne
0..*
Organisme
Nom organisme : {Unique}
Premier téléphone
Deuxième téléphone
{PARTITION}
0..1



CHAPITRE 2

Le modèle logique de données

OBJECTIFS

- Concepts fondamentaux liés à l'approche relationnelle pour le stockage et la manipulation des données dans un SGBD relationnel.
- Règles et principes assurant la conversion d'un modèle conceptuel en un modèle logique de données de type relationnel.
- Détail de la formulation d'un modèle relationnel de données dans la notation UML.
- Formes normales assurant la cohérence du modèle relationnel et leur lien avec les règles de modélisation conceptuelle vues au chapitre 1.

Ce chapitre décrit la deuxième étape dans la réalisation d'une base de données : l'élaboration d'un *modèle logique de données*. Comme nous l'indiquions en conclusion du chapitre précédent, ce chapitre ne vise pas l'exhaustivité. Nous le consacrons essentiellement aux principes d'élaboration

de modèles logiques de données de type relationnel, c'est à dire des modèles de données comportant des spécifications assurant son implantation à l'aide d'un SGBD relationnel.

Modèle logique de données ▸ Un modèle de données découlant d'un modèle conceptuel mais qui le raffine pour tenir compte des caractéristiques du type de SGBD utilisé pour la réalisation de la BD (*Logical Data model*).

Modèle relationnel de données ▸ Un modèle logique de données spécifiant un schéma pour une base de données relationnelle soit : les tables, les champs de chaque table et leurs propriétés, la clé primaire des tables, les clés étrangères assurant les liaisons entre les tables et les contraintes d'intégrité portant sur ces liaisons (*Relational Data model*).

En somme un modèle relationnel de données n'est qu'un cas particulier de modèle logique de données. Un modèle réseau de données ou un modèle hiérarchique de données font aussi partie des modèles de données de niveau logique. Considérant les objectifs particuliers de ce chapitre formulés plus haut, nous ne parlerons désormais que de *modèle relationnel de données* pour parler d'un modèle de données de niveau logique spécifiant le schéma d'une BD relationnelle.

Avant d'exposer les règles permettant de dériver un modèle relationnel de données à partir d'un modèle conceptuel de données exprimé à l'aide d'un diagramme entité-association, nous allons nous attarder aux origines et aux caractéristiques logiques des BD relationnelles.

ORIGINE ET TERMINOLOGIE DE L'APPROCHE RELATIONNELLE

Un peu d'histoire

On doit à E.F. Codd [COD 70], chercheur au laboratoire de recherche IBM de San Jose en Californie, la formulation des bases théoriques de l'approche relationnelle pour la réalisation des bases de données. La puissance de cette approche est qu'elle s'inscrit dans l'algèbre relationnelle, une théorie mathématique rigoureuse.

L'approche proposée par Codd couvre plusieurs aspects :

1. la structure de la base de données et les propriétés des champs où sont stockées les données ;
2. l'aspect de l'intégrité des données ;

3. l'aspect de manipulation des données par le biais d'opérateurs relationnels.

Le présent chapitre traite principalement du premier et du second aspect.

Les travaux de Codd ont donné naissance à ce qu'il est convenu d'appeler le modèle relationnel, ou l'approche relationnelle en matière de gestion de base de données, qui a donné lieu à un succès commercial remarquable. Les grands éditeurs de logiciels dont IBM, Oracle et Microsoft ont tour à tour adopté ce modèle pour la réalisation de leurs SGBD. Le domaine du logiciel libre n'est pas en reste puisque le SGBD le plus populaire sur la plate-forme Linux, My-SQL, fait appel au modèle relationnel et à son langage de manipulation de données : SQL.

Les fondements théoriques

Le modèle relationnel a ses racines dans l'algèbre relationnelle, une composante de la théorie des ensembles. Le terme *relation* réfère au concept mathématique suivant : considérant *n* ensembles E_1, E_2, … E_n, tout sous-ensemble du produit cartésien des *n* ensembles, noté $E_1 \times E_2 \times \ldots \times E_n$, constitue une relation.

Étant donné les deux ensembles $E_1 = \{a,b\}$ et $E_2 = \{c,d\}$, le produit cartésien de ces ensembles constitue l'ensemble de toutes les paires ordonnées dont le premier élément est un membre de E_1 et le deuxième, un membre de E_2, soit :

$E_1 \times E_2 = \{(a,c), (a,d), (b,c), (b,d)\}$

L'ensemble $R_1 = \{(a,c), (a,d)\}$ est un sous-ensemble de $E_1 \times E_2$ et il constitue une relation valide. Les éléments de la relation sont appelés des *tuples*. Les ensembles E_1, E_2, d'où sont tirées les valeurs placées dans un tuple, sont appelés les *domaines* de la relation. On dira par exemple que a et b appartiennent au *domaine* E_1, alors que c et d appartiennent au domaine E_2.

Toute l'approche relationnelle repose sur ces concepts. Ces structures élémentaires vont permettre par ailleurs de représenter des ensembles de données complexes. En vertu de cette approche, une relation dans une BD sera définie *en intention* en nommant la relation et ses attributs sans faire état des tuples qui forment explicitement la relation. Ainsi on peut définir une relation représentant l'ensemble des données conservées sur les employés d'une organisation de la manière suivante :

Employé[no_employé,nom_employé,prénom_employé,salaire_annuel]

Employé est le nom de la relation, **no_employé**, **nom_employé**, **prénom_employé**, et **salaire_annuel** en sont les attributs. L'attribut **no_employé** appartient au domaine constitué de tous les numéros d'employés acceptables. Ils pourraient être constitués de trois lettres et trois chiffres par exemple. Tous les autres attributs possèdent aussi leur domaine. L'attribut **salaire_annuel** appartient manifestement au domaine des nombres entiers positifs et non nuls exprimés avec un symbole monétaire.

Un attribut ou un groupe d'attributs de la relation permet par sa valeur de distinguer chaque tuple de la relation. On l'appelle la *clé primaire* de la relation. Les attributs qui forment la clé primaire sont soulignés dans la définition en intention de la relation. C'est le cas pour **no_employé** dans la relation **Employé**.

Une relation peut aussi être définie explicitement *en extension*. Dans cette définition, on fait voir à la fois le nom des attributs et les tuples qui constituent la relation. La figure 2-1 fait voir la relation Employé définie en extension. La clé primaire de la relation est mise en évidence avec des caractères gras.

FIGURE 2-1 **Relation *Employé* en extension**

Employé

no_employé	nom_employé	prénom_employé	salaire_annuel	
BLO001	Blouin	Paul	55 000 $	← Tuple
CAN020	Cantin	Sophie	58 000 $	
TRA034	Trahan	Pierre	60 000 $	

Puisque la définition la plus explicite d'une relation est sous forme d'une table, nous adopterons pour traiter des caractéristiques du modèle relationnel une terminologie basée sur la définition en extension des relations. Une autre raison qui justifie l'emploi de la terminologie qui suit est que l'utilisateur d'une BD relationnelle perçoit les données de la BD sous forme de tables.

Terminologie de l'approche relationnelle

Relation ▸ Une table comportant des lignes et des colonnes (*Relation*).

La table représente la vue logique d'une relation dans une BD relationnelle. La structure physique de stockage des données peut prendre des formes variées d'un SGBD à l'autre, selon les choix techniques faits par les concepteurs du SGBD.

Attribut ▸ Une colonne nommée de la table représentant la relation (*Attribute*).

Notons en passant que l'ordre des attributs n'a pas d'importance dans une relation. La même relation peut être représentée en inversant deux colonnes sans que cela ne constitue une nouvelle relation.

Domaine ▸ Ensemble des valeurs admises pour un attribut. Il établit les valeurs acceptables dans une colonne (*Domain*).

Une relation peut faire appel au même domaine pour plusieurs de ses attributs. La notion de domaine est fondamentale en matière de gestion de données. Elle permet d'exprimer des contraintes d'intégrité sémantique très fortes sur les données d'une BD.

Tuple ▸ Une ligne dans une table (*n-uplet*).

L'ordre des tuples dans la relation n'a pas d'importance. Une table, qu'elle soit triée ou non sur une colonne en ordre croissant de valeur, représente toujours la même relation.

La notion de tuple a été introduite au chapitre précédent pour parler d'une occurrence d'entité. Il s'agit sur le plan logique de la même notion. Comme nous le verrons plus loin en étudiant les règles de passage d'un modèle conceptuel à son pendant relationnel, une entité est traduite en table et, ce qui conceptuellement était considéré comme une occurrence d'entité, peut être représenté par une ligne dans une table.

Clé primaire ▸ Ensemble minimal de colonnes qui permet d'identifier de manière unique chaque tuple dans une table (*Primary key*).

Une clé primaire est souvent dite *simple*, si elle ne comporte qu'une seule colonne. S'il y a un risque, en faisant appel à une seule colonne, d'y trouver le cas échéant des doublons, cette clé ne peut être une clé primaire valide. Il est alors nécessaire de combiner la colonne à une autre, formant ainsi une clé primaire *composée* pour assurer l'identification unique des tuples. Une clé primaire composée peut être constituée de deux colonnes ou plus. Elle est quelque fois appelée clé *composite*.

Clé étrangère ▸ Une ou plusieurs colonnes dans une table qui a pour but d'assurer une liaison entre deux tables. On y arrive en dupliquant la clé primaire de la deuxième table dans la première. On l'appelle aussi clé externe (*Foreign key*).

Considérons les deux relations suivantes définies en intention :

Employé[no_employé,nom_employé,prénom_employé,salaire_annuel, no_département]

Département[no_département,nom_département,nb_employés]

L'attribut **no_département** apparaît dans les deux relations à la fois. Dans la relation **Département** il en est la clé primaire, Dans la relation **Employé**, il représente une clé étrangère visant à lier un tuple de **Employé** à un tuple de **Département**. Comme le montre la figure 2-2, cette liaison pourrait permettre à partir d'un tuple de **Employé** d'avoir accès à un tuple de **Département** correspondant et par ailleurs, partant d'un tuple de **Département** avoir accès à tous les tuples de **Employé**. Ces relations pourront permettre de faire état de tous les employés d'un même département.

FIGURE 2-2 **Relations *Employé* et *Département* en extension**

Employé

no_employé	nom_ employé	prénom_ employé	salaire_ annuel	no_ département
BLO001	Blouin	Paul	55 000 $	10
CAN020	Cantin	Sophie	58 000 $	20
TRA034	Trahan	Pierre	60 000 $	20

Clé étrangère

Département

no_département	nom_département	nb_employés
10	Comptabilité	13
20	Recherche et développement	8

Pour faire image, la table qui comporte une clé étrangère est souvent appelée une *table fille* en regard de la table d'où provient la clé étrangère, là où elle est clé primaire, que nous appellerons la *table mère*.

Base de données relationnelle ▸ Une collection de relations normalisées portant des noms distincts (*Relational Data Base*).

L'exigence de relations normalisées pour la cohérence d'une BD relationnelle est fondamentale. Comme nous le verrons à la fin de ce chapitre au moment de discuter de la normalisation des relations, normalisation qui vise essentiellement la cohérence et l'absence de redondance dans une BD, les règles de modélisation conceptuelle introduites au chapitre précédent visent la production d'un modèle relationnel de données dont les relations sont normalisées *de facto*.

La terminologie de l'approche relationnelle peut porter à confusion car selon les auteurs ou le contexte duquel est tiré un concept, il peut être exprimé avec des termes différents. Le tableau 2-1 tente de clarifier les termes employés et leurs correspondants, pas nécessairement leurs synonymes, selon le contexte où il est employé. Par exemple, les termes Relation, Table, Fichier et Entité type sont tous apparentés.

Bien utilisés, dans le contexte auquel le terme appartient, il ne devrait pas y avoir de confusion. Malheureusement, plusieurs auteurs font appel indifféremment à un terme ou à son correspondant sans égard au contexte où ils l'utilisent. Ceci est particulièrement flagrant dans la documentation qui accompagne les SGBD relationnels. Par exemple, dans le SGBD relationnel Access de Microsoft, on parle soit de ligne de table, soit d'enregistrement. Il est aussi question à la fois de colonne et de champ. On aurait pu très simplement s'en tenir aux deux seuls termes ligne et colonne. On y utilise aussi le terme Relation pour parler du lien entre deux tables par le biais d'une clé étrangère, ce qui est totalement inapproprié. Ces abus terminologiques conduisent non seulement à la confusion mais pire encore à la diffusion de faussetés. On a déjà lu dans un ouvrage portant sur MS Access que ce SGBD était de type relationnel car il permet d'établir des relations entre les tables !

TABLEAU 2-1 **Termes apparentés selon le contexte où on les utilise**

Mathématique	BD relationnelle	Gestion fichier	Conceptuel
Relation	Table	Fichier	Entité type
Tuple	Ligne	Enregistrement	Occurrence d'entité
Attribut	Nom de colonne	Champ	Attribut d'entité
Clé	Clé primaire	Clé enregistrement	Identifiant

LE MODÈLE RELATIONNEL DE DONNÉES : UNE REPRÉSENTATION GRAPHIQUE DU SCHÉMA DE LA BD

Le concepteur d'une BD relationnelle doit élaborer ce qu'il est convenu d'appeler le *schéma relationnel de la base de données*. Cette activité consiste à définir toutes les relations normalisées de la BD et les domaines de leurs attributs. Théoriquement cela consisterait à décrire par intention chaque relation, définir les domaines de tous les attributs et pour chaque attribut d'une relation, établir à quel domaine il appartient.

Les relations du schéma doivent toutes posséder les propriétés suivantes :

1. Une relation a un nom distinct de toutes les autres du même schéma
2. Chaque attribut d'une relation ne peut recevoir qu'une seule valeur atomique (type de données simple)
3. Chaque attribut a un nom distinct
4. Les valeurs d'un attribut font toutes partie du même domaine : même type de données et même contraintes d'intégrité
5. Chaque tuple de la relation est distinct ; pas de tuple en double
6. L'ordre des tuples n'a pas d'importance
7. L'ordre des attributs n'a pas d'importance

En pratique on arrive à réaliser un schéma relationnel en élaborant une représentation graphique du schéma qui tient lieu de *modèle relationnel de données* de la BD. Celle-ci est effectuée de préférence à l'aide d'un outil logiciel qui permettra au concepteur de valider son modèle, notamment d'assurer le respect des propriétés des relations données plus haut.

Le modèle relationnel de données est une représentation du schéma de la BD beaucoup moins lourde qu'une représentation des relations en intention tout en étant tout aussi expressive. Or cette représentation graphique peut être dérivée directement du modèle conceptuel de la BD en appliquant un certain nombre de règles de dérivation assurant le passage du niveau conceptuel au niveau logique. Dériver le modèle relationnel directement d'un modèle conceptuel comporte de nombreux avantages. Non seulement la dérivation peut-elle être automatisée grâce à un outil logiciel approprié, mais les règles de modélisation conceptuelle décrites au chapitre 1 assurent le respect de la plupart des propriétés des relations évoquées ci-haut. Ainsi la règle de non-redondance assure les propriétés 1 et 3. La règle de description permet le respect de la propriété 2. Les contraintes de domaine garantissent la propriété 4. La propriété 5 relève de la règle d'identité.

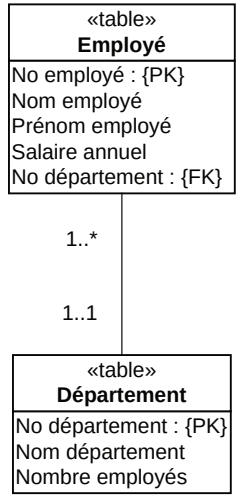
Nous traitons dans cette section de dérivation d'un modèle relationnel de données à partir d'un modèle conceptuel de données par une étude rigoureuse des règles de dérivation. Pour ce faire, nous présentons les conventions de représentation d'un modèle relationnel de données formulé à l'aide de la notation UML.

Notation UML et modèle relationnel de données

Tout comme pour le modèle conceptuel, le modèle relationnel est formulé par le biais du diagramme de classes de UML. Ici encore on mise sur le concept de *prototype* du langage UML pour donner au diagramme de classes et à la case rectangulaire une sémantique particulière.

Tout comme pour le modèle conceptuel, la case rectangulaire d'un modèle relationnel porte une mention de prototype. Cette fois il s'agit de la mention *«table»* comme le montre la figure 2-3 où est présentée la version graphique du schéma relationnel de la figure 2-2. Dans un modèle relationnel, les associations **SONT TOUTES DES ASSOCIATIONS BINAIRES** portant des contraintes de multiplicité entre des tables, d'où la présence de la mention *«table»* en guise de prototype. Lorsqu'il est clair que le modèle présenté est un modèle relationnel, cette mention peut être masquée. Nous adopterons cette convention tout au long de ce chapitre pour alléger les modèles.

FIGURE 2-3 **Modèle relationnel correspondant au schéma de la figure 2-2**



Dans un modèle relationnel les associations entre les tables n'ont pas à être nommées. Elles indiquent simplement qu'il existe un lien entre les deux tables et que la clé primaire d'une table marquée {PK} (pour *Primary Key*) est reproduite dans l'autre table sous forme de clé étrangère. Une clé étrangère est marquée {FK} (pour *Foreign Key*).

La présence d'un attribut avec la mention {PK} dans une table reflète une contrainte d'intégrité appelée *contrainte d'intégrité d'entité*. Nous en avons déjà fait état au chapitre 1 et donnons ici une interprétation apparentée tirée du contexte de l'approche relationnelle.

Intégrité d'identité ▸ Contrainte appliquée à une clé primaire qui assure qu'aucun attribut d'une clé primaire ne peut être nul (*Entity integrity*).

Un corollaire de cette contrainte d'intégrité est que la mention {PK} devrait toujours être suivie de la contrainte {Non nul} dans un modèle relationnel. Encore une fois, puisque {PK} sous-entend {Non nul}, on inscrit rarement les deux mentions dans un modèle relationnel de manière à alléger le diagramme.

Toutes les tables devraient posséder une clé primaire et les associations entre les tables assurées par ces clés seront de deux types :

1. les multiplicités maximales sont 1 d'une part et 1 de l'autre, on parle ici d'une association ***un à un*** ;
2. les multiplicités maximales sont 1 d'une part et * de l'autre, on parle d'une association ***un à plusieurs***, c'est le cas du modèle 2-3.

Contrairement au modèle conceptuel, le modèle relationnel ne supporte que les associations binaires ***un à un*** et ***un à plusieurs*** donc aucune association ***plusieurs à plusieurs*** n'est acceptable dans ce dernier type de modèle. Il s'agit d'une erreur grave sur le plan sémantique d'avoir des multiplicités maximales * de part et d'autre d'une association dans un modèle relationnel. La raison est simple : comme un attribut clé étrangère ne peut avoir qu'une seule valeur dans une ligne donnée de la table, elle ne peut assurer la liaison de cette ligne qu'à une seule autre ligne de la deuxième table. La multiplicité maximale est donc systématiquement à 1 du côté de la table possédant la clé primaire qui a été copiée dans l'autre table avec la mention {FK} (clé étrangère).

Pour revenir à l'analogie *mère-fille*, on peut facilement résumer le concept de la manière suivante : une ligne dans une table ne peut avoir qu'une seule mère provenant d'une autre table. La valeur de la clé primaire de sa ligne mère doit être présente sous forme d'attribut de catégorie clé étrangère dans la ligne fille.

Les multiplicités ont la même sémantique que sur le plan conceptuel, à la différence que le modèle relationnel fait état du nombre de lignes (ou tuples) d'une table qui sont liées à une ligne (ou un tuple) de l'autre table.

Un modèle relationnel complet devrait notamment faire état du domaine de chaque attribut. Cela est habituellement réalisé en inscrivant à la suite des attributs et des mentions {PK} et {FK} le cas échéant, le type de données de l'attribut et les contraintes d'intégrité de l'attribut. Par exemple, si la clé primaire porte sur un seul attribut, il s'agit donc d'une clé simple. Cet attribut devrait avoir les contraintes d'intégrité {Unique} et {Non nul}. Rappelons qu'une clé primaire est par définition obligatoire et non redondante.

Si le concepteur a pris soin d'inscrire le type de données des attributs et les contraintes d'intégrité sémantique dans le modèle conceptuel, via le dictionnaire de données d'un logiciel de modélisation, ces propriétés des attributs peuvent être reprises tel quel dans le modèle relationnel dérivé.

RÈGLES DE DÉRIVATION DES RELATIONS À PARTIR D'UN MODÈLE CONCEPTUEL DE DONNÉES

Le cas des entités

Pour chaque entité du modèle conceptuel, une table est créée dans le modèle relationnel, sauf dans le cas d'une association d'héritage où les attributs des entités en cause peuvent être combinés pour former une table. Les attributs de l'entité deviennent les attributs de la table et, dans le cas d'une entité forte, son identifiant, simple ou composé, devient la clé primaire de la table. Que la clé primaire soit simple ou composée, chaque attribut formant la clé porte la mention {PK}.

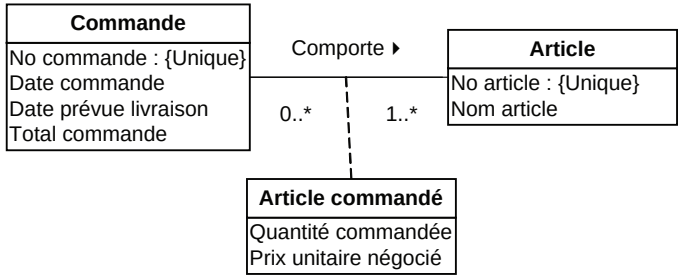
La clé primaire d'une entité faible est basée sur son identifiant implicite. Rappelons que l'on retrouve deux types d'entités faibles dans un modèle conceptuel : les entités d'association et les entités représentant un composant d'une entité composite.

Dérivation à partir d'une entité d'association

En ce qui concerne une entité d'association, la table correspondante aura comme clé primaire une clé obligatoirement composée des attributs de tous les identifiants des entités participant à l'association et chaque attribut de la clé composée portera la mention {PK, FK}.

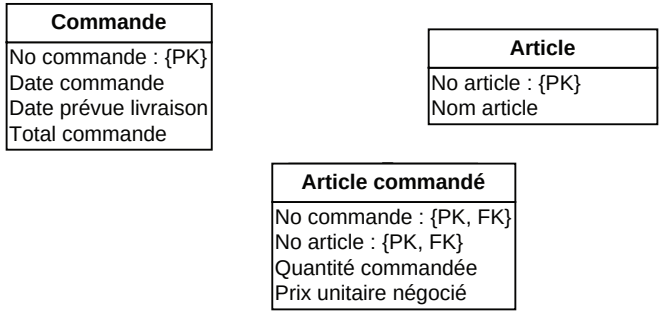
Illustrons d'un exemple cette première règle de dérivation. La figure 2-4 comporte trois entités dont une entité d'association, **Article commandé**.

FIGURE 2-4 **Modèle conceptuel comportant une entité d'association**



Le modèle relationnel dérivé comporte trois tables dont une table, **Article commandé**, possède une clé primaire composée obligatoirement des clés primaires des deux autres tables. Comme les attributs de la clé sont repris des autres tables, ils doivent être considérés comme des clés étrangères. Rien n'interdit dans le modèle relationnel qu'un attribut soit à la fois une clé étrangère tout en constituant la clé primaire, en tout ou en partie.

FIGURE 2-5 **Ébauche du modèle relationnel dérivé du modèle 2-4**

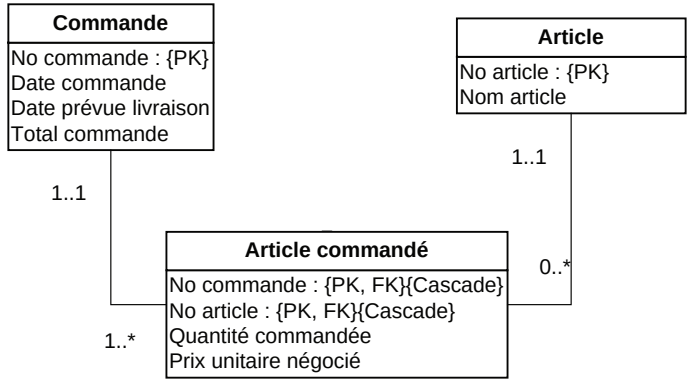


Cet exemple comporte des entités associées, **Commande** et **Article**, qui possèdent chacune un identifiant simple. Si ces entités devaient avoir un identifiant composé, par exemple de deux attributs, les tables dérivées, **Commande** et **Article**, auraient aussi une clé composée de deux attributs. Quant à la table dérivée de l'entité d'association, elle devrait avoir une clé primaire composée de quatre attributs ! On voit là toute l'importance de choisir des identifiants simples pour les entités du modèle conceptuel afin d'éviter de dériver des clés primaires complexes dans le modèle relationnel.

Le modèle relationnel de la figure 2-5 ne saurait être complet sans qu'il fasse état explicitement des associations entre les tables et de leurs multiplicités. La table dérivée de l'entité d'association, une table fille, doit être liée à toutes les tables dérivées des autres entités associées qui agiront comme tables mères.

Chaque association portera une multiplicité 1..1 du côté de la table mère et quant aux multiplicités du côté de la table fille, elles sont reprises du modèle conceptuel. Ainsi dans le modèle 2-4, on note la multiplicité 0..* sur la terminaison d'arrivée de l'association lue à partir de l'entité **Article**. La multiplicité sera donc 0..* du côté de la table fille **Article commandé** qui est maintenant la terminaison d'arrivée de l'association lue à partir de la table **Article**. Par ailleurs on a la multiplicité 1..* sur la terminaison d'arrivée de l'association lue à partir de l'entité **Commande**. La multiplicité sera donc 1..* du côté de la table fille **Article commandé** qui est maintenant la terminaison d'arrivée de l'association lue à partir de la table **Commande**. La figure 2-6 donne le modèle relationnel complet dérivé de 2-4.

FIGURE 2-6 **Modèle relationnel dérivé du modèle 2-4**



Le lecteur aura noté la présence de la mention {Cascade} à la suite des clés étrangères présentes dans la table dérivée de l'entité d'association. Cette mention fait état d'une contrainte d'intégrité fort importante appelée *contrainte d'intégrité référentielle*.

Intégrité référentielle ▸ Contrainte appliquée à une clé étrangère présente dans une table stipulant que la valeur de cette clé peut être nulle, ou si elle ne l'est pas, elle doit correspondre à la valeur d'une clé primaire d'une ligne de la table mère (*Referential integrity*).

L'intégrité référentielle signifie sur le plan opérationnel qu'une ligne ne peut être ajoutée à une table fille à moins que la valeur de la clé étrangère soit nulle ou qu'elle soit déjà présente dans la clé primaire d'une ligne de la table mère (***contrainte en ajout***). La mention {Cascade} comporte des restrictions supplémentaires :

1. si une ligne de la table mère est supprimée, toutes les lignes associées à celle-ci via la clé étrangère dans la table fille sont aussi supprimées (***contrainte de suppression en cascade***) ;
2. si la valeur de la clé primaire d'une ligne est modifiée, les valeurs des clés étrangères des lignes associées dans la table fille sont modifiées pour continuer d'assurer la liaison (***contrainte de mise à jour en cascade***).

Ces contraintes sont incontournables dans le cas d'une entité faible car son existence dépend de l'existence d'une entité forte. Logiquement toute occurrence d'entité forte qui est détruite entraîne avec elle la disparition des entités faibles qui en dépendent.

Simplification de la clé primaire de la table dérivée d'une entité d'association. S'il existe une multiplicité maximale de 1 sur l'arc d'une association mettant en cause une entité d'association, la clé primaire de la table fille dérivée de l'entité d'association peut être simplifiée. En effet, la clé primaire de la table mère du côté de la multiplicité maximale 1 peut être retirée de la clé primaire de la table fille tout en demeurant une clé étrangère.

Considérons à nouveau le modèle conceptuel de la figure 2-4. Apportons y une contrainte selon laquelle une commande n'aurait qu'un et un seul article à la fois. Sur l'association **Comporte**, la multiplicité du côté de l'entité **Article** serait alors 1..1. La table fille dérivée, **Article commandé**, pourrait avoir comme clé primaire un seul attribut, **No commande**, mais **No article** devrait tout de même y être présent à titre de clé étrangère mais cette fois SANS la mention {Cascade}.

La démarche de dérivation préconisée pour une entité d'association appartenant à une association binaire peut facilement être étendue à une entité d'association appartenant à une association de degré supérieur. Les principes de dérivation demeurent globalement les mêmes. Nous aurons l'occasion d'étudier les quelques différences à la rubrique traitant de la dérivation des associations de degré supérieur.

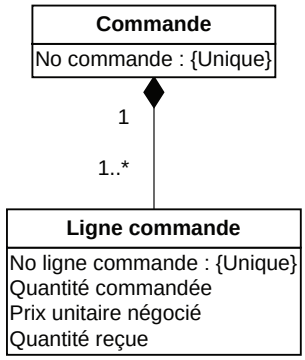
Dérivation à partir des entités d'une composition

Un composant lié à une entité composite a un identifiant partiellement défini par son composite. En effet, elle possède son propre identifiant qui garantit une identification unique des occurrences d'un même composite

mais il ne peut assurer une identification unique des toutes les occurrences de l'entité en général. Pour ce faire, son identifiant doit être combiné à l'identifiant du composite.

Avec pour conséquence qu'une table sera dérivée du composite et une autre table sera dérivée du composant. Cette dernière aura comme clé primaire la combinaison de l'identifiant du composite marqué {PK} et de l'identifiant propre au composant marqué {PK, FK}. Quant à la table dérivée du composite, sa clé primaire est l'identifiant de l'entité composite comme pour toute entité forte.

FIGURE 2-7 **Modèle conceptuel comportant une composition**

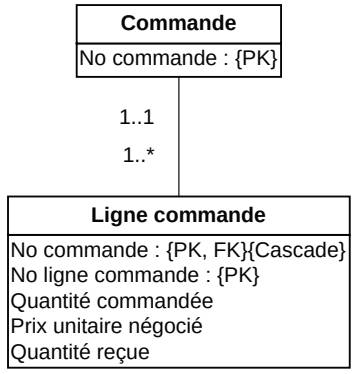


La figure 2-7 montre une association de composition où l'identifiant implicite du composant, ici **Ligne commande**, sera formé de son propre identifiant combiné à celui de son composite.

Le modèle relationnel dérivé doit montrer explicitement les clés primaires des tables. D'où la présence d'une clé primaire composée pour la table **Ligne commande** où l'attribut **No commande** porte la mention {PK, FK} indiquant par là qu'il s'agit d'un élément de la clé primaire tout en étant une clé étrangère comme l'illustre la figure 2-8.

De plus, considérant la dépendance existentielle de **Ligne commande** envers **Commande**, il est absolument nécessaire d'assurer l'intégrité référentielle entre les deux tables ainsi que la suppression et la mise à jour en cascade des lignes de la table **Ligne commande** par le biais de la clé étrangère **No commande**. Quant aux multiplicités, elles sont reprises intégralement à partir du modèle conceptuel. Rappelons que la multiplicité 1 prise seule, signifie de fait 1..1.

FIGURE 2-8 **Modèle relationnel dérivé de la figure 2-7**



La dérivation des tables et de leur clé primaire est la première étape du processus de passage d'un modèle conceptuel au modèle relationnel correspondant. L'autre étape consiste à analyser les associations et à dériver le cas échéant d'autres tables et d'autres clés étrangères selon la nature des associations.

Les associations binaires

Nous allons étudier sous cette rubrique la mise en œuvre des associations binaires dans le modèle relationnel sur la base de leurs multiplicités maximales et traiterons du cas singulier de l'association réflexive. Les contraintes de multiplicité qu'impose le modèle conceptuel doivent se traduire par des contraintes d'intégrité appliquées à la clé étrangère assurant la liaison entre les deux tables d'une association binaire.

Les règles de dérivation qui suivent ne s'appliquent qu'aux associations binaires ne comportant pas d'entité d'association. Le cas des associations binaires comportant une entité d'association a été traité plus haut. Les règles de dérivation des associations binaires visent à déterminer dans quelle table sera placée la clé étrangère incarnant la liaison. Dans tous les cas, les multiplicités de l'association au modèle conceptuel sont reprises intégralement sur l'association entre les tables dérivées.

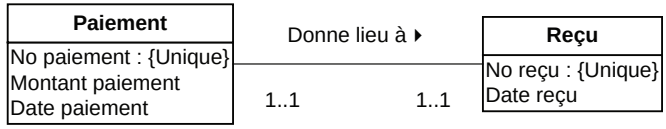
Association binaire un à un

Dans un tel cas, la dérivation de la clé étrangère repose sur l'analyse des multiplicités minimales de l'association. Cela consiste à établir si la participation des occurrences à l'association est obligatoire d'un côté ou de l'autre ou des deux à la fois.

Si la participation est obligatoire des deux côtés de l'association (multiplicités minimales 1 de part et d'autre), l'une ou l'autre des tables dérivées des entités peut agir comme table *mère*. Cependant, si une seule des deux entités est associée à une troisième, on choisira comme table *fille* celle dérivée de l'entité qui comporte une deuxième association. Si toutes deux sont associées à une autre, on choisit comme table fille celle qui possède une clé primaire composée.

Considérons la figure 2-9 où une association binaire montre des multiplicités 1..1 de part et d'autre des entités concernées.

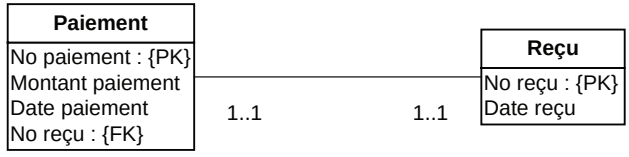
FIGURE 2-9 **Association binaire *un à un* avec participation obligatoire de part et d'autre**



Pour ce modèle, nous faisons l'hypothèse que l'entité **Paiement** est associée par ailleurs à l'entité **Client** (non illustrée). La table *fille* sera donc **Paiement**, la table *mère* sera **Reçu**, et la clé primaire de la table *mère*, **No reçu**, est reproduite dans la table *fille* à titre de clé étrangère (figure 2-10). Les multiplicités sont reprises telles quelles du modèle conceptuel.

Sans cette hypothèse, la table *fille* aurait tout aussi bien pu être **Reçu**.

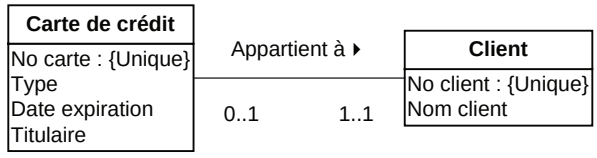
FIGURE 2-10 **Table dérivée du modèle 2-9**



Si la participation est optionnelle sur au moins un côté de l'association (multiplicité minimale 0 d'un côté ou de l'autre), la table dérivée de l'entité qui a la participation optionnelle devient la table fille. Une copie de la clé primaire de la table *mère* est déposée dans la table *fille* à titre de clé étrangère.

La figure 2-11 montre un modèle conceptuel où une carte de crédit ne peut être associée qu'à un et un seul client, mais par ailleurs on associe au plus une seule carte de crédit au client pour fin de paiement.

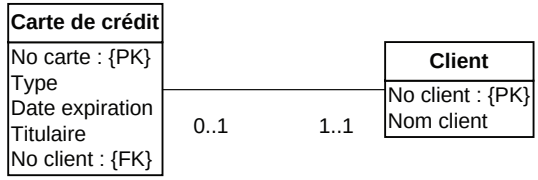
FIGURE 2-11 **Modèle conceptuel avec association *un à un*, avec une participation optionnelle**



L'association *un à un* étant optionnelle côté **Carte de crédit**, la table dérivée de cette entité est considérée comme table *fille* et la clé primaire de la table *mère* **Client**, soit **No client**, est copiée dans la table *fille* comme clé étrangère.

La figure 2-12 montre le modèle relationnel dérivé du modèle conceptuel 2-11. On note que les multiplicités montrées sur l'association entre les tables sont les mêmes que celles du modèle conceptuel.

FIGURE 2-12 **Modèle relationnel dérivé de 2-11**



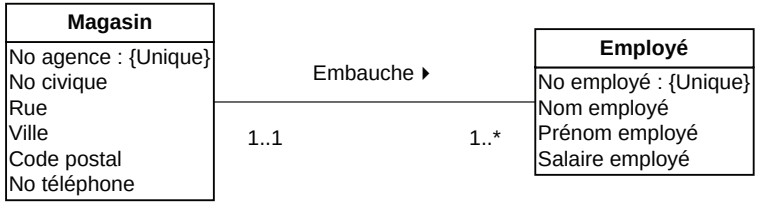
Si la participation est optionnelle de part et d'autre d'une association *un à un*, l'une ou l'autre des tables peut être arbitrairement considérée comme table *fille*, mais on retiendra en priorité comme table fille celle des deux qui fait l'objet d'une autre association.

Association binaire un à plusieurs

Les associations *un à plusieurs* sont les plus nombreuses dans un modèle conceptuel. Or il s'avère que c'est aussi le type d'association comportant la règle de dérivation la plus simple. Elle se résume ainsi : la table dérivée de l'entité du côté de la multiplicité maximale plusieurs (*****) est considérée la table *fille*. En conséquence, une copie de la clé primaire de la deuxième table, la table *mère*, est déposée dans la table *fille* à titre de clé étrangère.

Cette règle ne comporte aucune exception.

FIGURE 2-13 **Modèle conceptuel avec association binaire *un à plusieurs***



La figure 2-13 montre un modèle conceptuel avec une association binaire *un à plusieurs* où la participation est obligatoire de part et d'autre. La participation ne joue aucun rôle dans la dérivation du modèle relationnel pour ce type d'association. La table dérivée de **Employé** devient la table *fille*, car elle est côté plusieurs, et la table **Magasin**, la table *mère*. La clé primaire de **Magasin**, **No agence**, devra apparaître dans **Employé** sous forme de clé étrangère. La figure 2-14 présente le modèle relationnel dérivé.

FIGURE 2-14 **Modèle relationnel dérivé de 2-13**

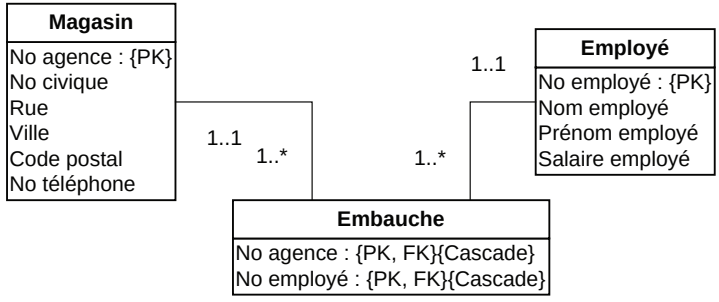
Association binaire plusieurs à plusieurs

Une association *plusieurs à plusieurs* sans entité d'association donne lieu à une table fille comportant comme clé primaire la combinaison des clés primaires des tables mères. Chaque composant de la clé primaire est aussi clé

étrangère. Aucun autre attribut n'est présent dans la table. Deux tables mères sont présentes et deux associations vont assurer la liaison avec leur table fille. Chaque association portera une multiplicité 1..1 du côté de la table mère et quant aux multiplicités du côté de la table fille, elles sont reprises du modèle conceptuel comme pour une entité d'association. S'il y a au modèle conceptuel la multiplicité 0..* sur la terminaison d'arrivée de l'association lue de **A** vers **B**, il y aura multiplicité 0..* du côté de la table fille pour l'association la liant à **A**. S'il y a au modèle conceptuel la multiplicité 1..* sur la terminaison d'arrivée de l'association lue de **B** vers **A**, il y aura multiplicité 1..* du côté de la table fille pour l'association la liant à **B**.

Si nous voulons que le modèle de la figure 2-13 reflète un contexte plus général et moins restrictif où un employé peut être embauché dans plusieurs magasins, nous aurions alors une association binaire entre **Magasin** et **Employé** du type *plusieurs à plusieurs*. Le modèle relationnel résultant, figure 2-15, comporterait alors une nouvelle table qui porterait le nom de l'association et dont les attributs se résument à la combinaison des clés primaires des tables mères. Chaque attribut est une clé étrangère pour laquelle une contrainte d'intégrité en ajout, suppression et mise à jour devra être appliquée.

FIGURE 2-15 **Modèle relationnel dérivé d'une association binaire *plusieurs à plusieurs* dans le modèle 2-13**



Il y a un doute sur le fait que la clé primaire de la table fille dérivée d'une association binaire PLUSIEURS À PLUSIEURS exclue tout doublon. Il n'existe aucune assurance que la clé primaire composée de la table fille dérivée d'une association *plusieurs à plusieurs* sans entité d'association soit valide. Tout dépend du contexte. La table **Embauche** du modèle relationnel 2-15 ne peut conserver qu'une ligne pour un employé donné dans un magasin donné. Si on accepte qu'un employé puisse être embauché plus d'une fois dans le même magasin, la clé primaire composée de **No agence** et **No employé** est insuffisante. Elle peut corresponde à plusieurs lignes de la

table **Embauche** ce qui ne peut rendre la clé primaire non valide. En cas de doute sur l'unicité d'accès à une ligne dans la table dérivée, il y a lieu d'appliquer une technique d'optimisation sur la clé primaire.

Remplacement d'une clé composée par une clé primaire simple à génération automatique de valeur : Si le modélisateur doute de l'unicité d'accès par la clé primaire de la table dérivée d'une association binaire PLUSIEURS À PLUSIEURS sans entité d'association ou d'une association de degré supérieur sans entité d'association ne comportant que des multiplicités maximales PLUSIEURS, il peut donner à la table fille dérivée une clé primaire simple artificielle. La clé primaire portera le nom de la table fille précédé de **No**. Cette clé primaire simple portera la mention {PK auto} signifiant par là que le SGBD générera automatiquement, lors de l'ajout d'une nouvelle ligne dans la table, une valeur différente pour la clé primaire. Tous les SGBD relationnels disposent d'un mécanisme pour réaliser cette fonction fort utile. Comme nous le verrons plus loin, cette technique peut aussi bien être appliquée, en toute généralité, aux tables comportant des clés primaires composées pour optimiser le modèle relationnel.

Revenons au modèle relationnel 2-15. Une clé primaire artificielle **No embauche** devrait être créée avec la mention {PK auto}. Quant aux attributs qui constituaient la clé primaire composée, ils perdent la mention PK mais demeurent dans la table à titre de clés étrangères et porteront en conséquence la mention {FK}{Cascade}.

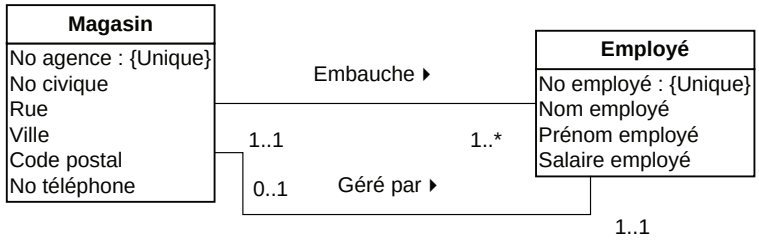
Priorité d'application des règles de dérivation

Qu'arrive-t-il si les deux mêmes entités sont associées avec des associations de natures différentes ? Par exemple à la fois par une association du type *un à un* et une autre de type *un à plusieurs*.

Réponse : il n'y a pas de priorité d'application des règles. CHAQUE ASSOCIATION DONNE LIEU À LA CRÉATION DE CLÉS ÉTRANGÈRES, DANS LA MÊME TABLE OU DANS LES DEUX TABLES. Seule restriction : si les deux même entités sont liées par plus d'une association, il ne peut y avoir fusion de leurs attributs dans une seule table même si l'une d'elle est du type *un à un* avec participation obligatoire de part et d'autre.

Le modèle de la figure 2-16 comporte deux associations sur les mêmes entités, une première *un à un* et une autre *un à plusieurs*. Les deux associations doivent être implantées par des clés étrangères distinctes dans le modèle relationnel.

FIGURE 2-16 **Plus d'une association impliquant les mêmes entités**

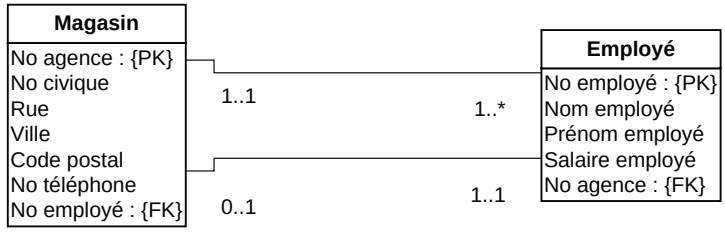


La table dérivée de **Magasin** est la table *fille* pour l'association **Géré par** qui est du type *un à un*. La règle veut en effet que dans le cas d'une association *un à un*, la table du côté de la multiplicité minimale 0 (optionnel) soit la table *fille*. La clé primaire de **Employé**, soit **No employé**, doit donc s'y trouver comme clé étrangère.

La table **Employé** est par ailleurs une table *fille* pour l'autre association car elle dérive d'une entité liée à l'association *un à plusieurs* **Embauche** et qu'elle est placée du côté plusieurs de celle-ci. La clé primaire de **Magasin**, **No agence**, doit en conséquence s'y trouver à titre de clé étrangère.

La figure 2-17 montre le modèle relationnel dérivé de 2-16 après application des règles spécifiques à chaque type d'association.

FIGURE 2-17 **Modèle dérivé d'associations impliquant les mêmes entités**



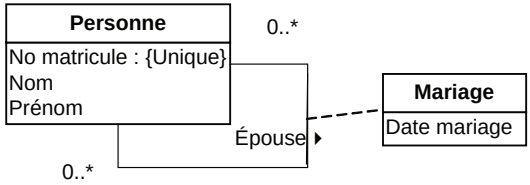
Association réflexive

La dérivation d'une association réflexive reprend les règles évoquées plus haut en les spécialisant pour tenir compte du fait qu'une entité est associée à elle-même.

Cas 1: L'association réflexive comporte une entité d'association.
En vertu des règles données plus tôt, l'entité d'association donne lieu à une table fille associée à la table mère dérivée de la seule entité en cause. La table dérivée de l'entité d'association comporte les attributs de cette dernière avec comme clé primaire deux exemplaires de la clé primaire de la table mère, les exemplaires portent un nom différent pour éviter toute redondance. Chaque exemplaire est aussi une clé étrangère avec la mention {Cascade} pour assurer l'intégrité référentielle.

Considérons le modèle conceptuel de la figure 2-18.

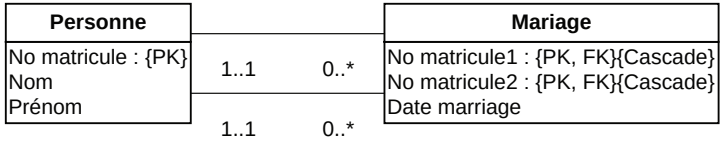
FIGURE 2-18 **Modèle conceptuel avec association réflexive et entité d'association**



Ce modèle permet de faire état de tous les mariages qu'une personne peut avoir contractés. On fait l'hypothèse qu'une personne ne peut épouser la même personne qu'une seule fois. Chaque mariage peut donc être identifié de manière unique par la combinaison des matricules des deux personnes qui s'épousent à la date donnée.

Le résultat de la dérivation du modèle relationnel est donné à la figure 2-19. La table fille **Mariage** est dérivée de l'entité d'association. Elle comporte une clé primaire composée des deux numéros matricules des personnes contractant un mariage. Chaque matricule porte un nom d'attribut différent. À chaque matricule correspond une association reprenant du côté de la table fille les multiplicités du modèle conceptuel. Le matricule est aussi une clé étrangère pour laquelle l'intégrité référentielle doit être assurée en ajout, en suppression et en mise à jour.

FIGURE 2-19 **Modèle relationnel dérivé de 2-18**



Cas 2 : L'association réflexive est du type *un à un* ou *un à plusieurs*.

Il s'agit d'une situation où l'association réflexive ne porte pas d'entité d'association et où les multiplicités maximales sont de 1 d'un côté ou de l'autre de l'association, ou bien des deux côtés à la fois. Dans ce cas une seule table est dérivée et elle comporte une clé étrangère. Celle-ci est un double de sa clé primaire qui doit porter un nom différent. Les multiplicités de l'association au modèle conceptuel sont reprises intégralement dans le modèle relationnel dérivé.

La figure 2-20 est aussi un modèle portant sur la notion de Mariage où cette fois on ne fait état que du dernier mariage contracté par une personne. En effet, on voit clairement qu'une personne contracte au plus un mariage et la date de ce mariage peut être considérée comme un attribut de l'entité personne. Cette association réflexive est du type *un à un*.

FIGURE 2-20 **Modèle conceptuel, sans entité d'association, avec association réflexive *un à un***

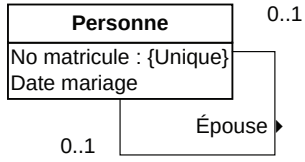
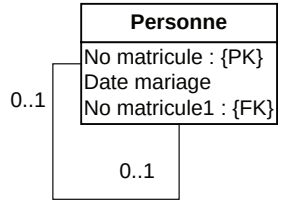


FIGURE 2-21 **Modèle relationnel dérivé du modèle conceptuel 2-20**



La figure 2-22 fait état d'une association réflexive du type *un à plusieurs*. Un employé peut diriger d'autres employés et un employé n'a qu'un et un seul directeur.

Les mêmes règles s'appliquent pour une association réflexive *un à un*. On note à la figure 2-23 que les multiplicités du modèle relationnel dérivé sont en tout point conformes à celles du modèle conceptuel.

FIGURE 2-22 **Modèle conceptuel, sans entité d'association, avec association réflexive *un à plusieurs***

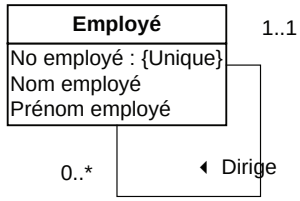
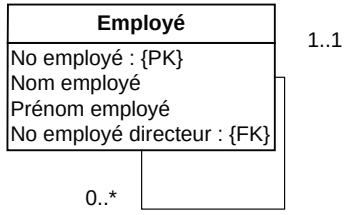


FIGURE 2-23 **Modèle relationnel dérivé du modèle conceptuel 2-22**

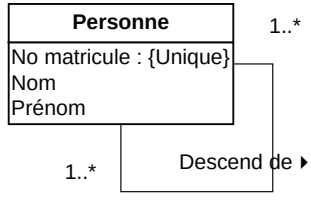


Cas 3 : L'association réflexive est du type *plusieurs à plusieurs*.

Une association réflexive *plusieurs à plusieurs* est traitée comme une association réflexive comportant une entité d'association. Une table doit être créée pour assurer une liaison plusieurs à plusieurs sur la même table. Cette table n'a pas d'attribut propre. Sa clé primaire est une combinaison de deux exemplaires de la clé primaire de la table dérivée de l'entité. Chaque exemplaire aura un nom différent et sera traité comme une clé étrangère avec contrainte d'intégrité référentielle en ajout, suppression et mise à jour (mention {Cascade}).

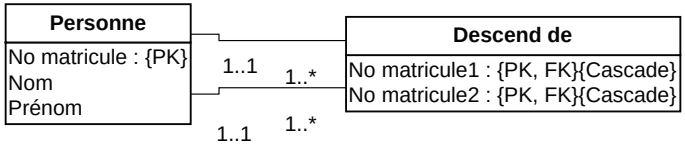
La figure 2-24 modélise la descendance d'une personne. Une personne descend de plusieurs personnes et par ailleurs possède plusieurs descendants.

FIGURE 2-24 **Modèle conceptuel, sans entité d'association, avec association réflexive *plusieurs à plusieurs***



Le modèle relationnel dérivé (figure 2-25) fait état d'une table jouant le rôle d'une association *plusieurs à plusieurs* et le nom de la table reprend le nom de l'association du modèle conceptuel. Comme pour une table dérivée d'une entité d'association, la multiplicité 1..1 sera toujours présente du côté de la table mère. Du côté fille, les multiplicités du modèle conceptuel sont reprises. Elles valent toutes deux 1..* dans cet exemple mais pourraient fort bien être différentes si elles étaient différentes du modèle conceptuel.

FIGURE 2-25 **Modèle relationnel dérivé de 2-24**



Les associations de degré supérieur

Toute association de degré supérieur donne lieu à une table qui s'ajoute aux tables dérivées des entités associées.

S'il s'agit d'une association de degré supérieur qui comporte une entité d'association, les règles vues précédemment pour la conversion d'une entité d'association présente sur une association binaire s'appliquent :

- chaque entité associée devient une table mère ;
- l'entité d'association devient une table fille ;
- la clé primaire de la table fille combine les clés primaires des tables dérivées des tables mères associées et chaque élément de la clé est une clé étrangère ;
- cependant, si une des multiplicités maximales sur les arcs vaut *un* (1), la clé primaire de la table fille est la combinaison des clés primaires des tables associées sauf celle associée par l'arc retenu où une multiplicité maximale 1 est présente ;
- les clés primaires des tables mères qui n'ont pas été intégrées à la clé primaire sont présentes à titre de clés étrangères dans la table fille et les attributs de l'entité d'association s'y ajoutent. L'intégrité référentielle en ajout, suppression et mise à jour NE S'APPLIQUE QU'AUX CLÉS ÉTRANGÈRES QUI FONT PARTIE DE LA CLÉ PRIMAIRE.
- les multiplicités sont toutes à 1..1 du côté des tables mères.

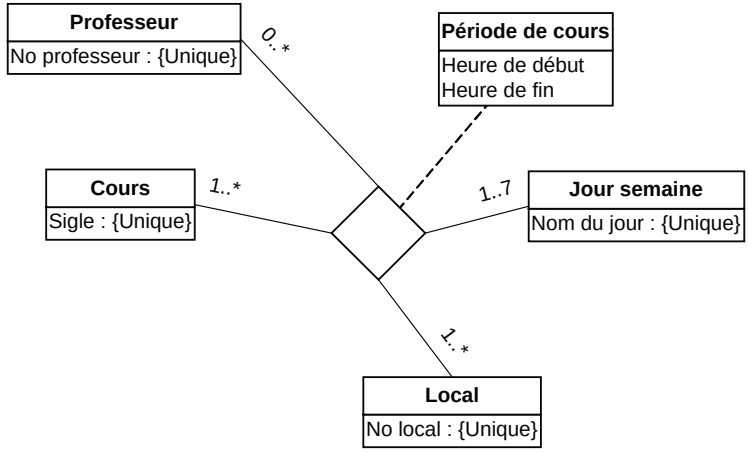
Une règle importante QUI NE S'APPLIQUE qu'aux associations de degré supérieur et non pas aux associations binaires avec entité d'association :

- Les multiplicités sont toutes à 0..* du côté de la table fille

Illustration 1 : L'association de degré supérieur comporte une entité d'association.

Nous illustrons ici la dérivation d'une entité d'association présente sur une association de degré supérieur. La figure 2-26 montre une association de degré supérieur où toutes les multiplicités maximales sont *plusieurs*. La table fille **Période de cours** doit en conséquence posséder une clé primaire composée des clés primaires des quatre tables mères.

FIGURE 2-26 **Modèle conceptuel avec entité d'association sur une association de degré supérieur**



Le modèle relationnel de la figure 2-27 est dérivé du modèle conceptuel de la figure 2-26. Il comporte cinq tables, dont une table dérivée de l'entité d'association qui possède une clé primaire composée des clés primaires des quatre autres. Cette table, **Période de cours**, est une table fille.

Les clés primaires des quatre tables mères qui sont copiées dans la table fille sont toutes des attributs considérés comme clés étrangères sur lesquelles la contrainte d'intégrité référentielle s'applique. Les attributs propres à l'entité d'association, **Heure de début** et **Heure de fin**, apparaissent à leur suite.

Les multiplicités placées du côté de la table fille sont toutes 0..*. Les multiplicités du côté des tables mères sont systématiquement 1..1. Puisque toutes les associations comportent des multiplicités maximales *plusieurs* du côté de la table fille, sa clé primaire ne peut être simplifiée.

FIGURE 2-27 **Modèle relationnel dérivé du modèle 2-26**

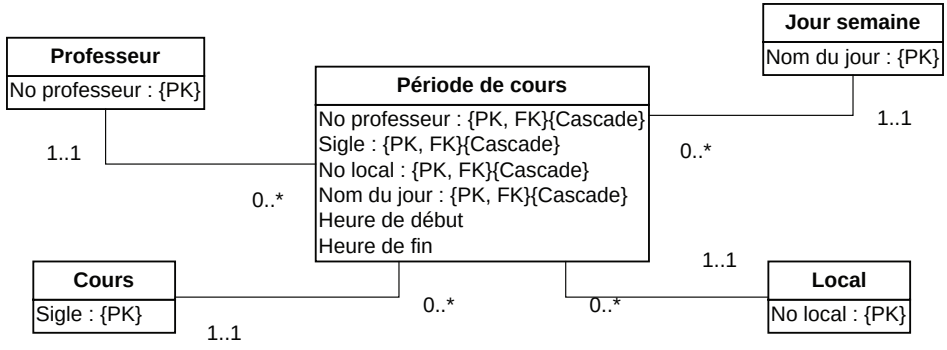


Illustration 2 : L'association de degré supérieur ne comporte pas d'entité d'association.

Si l'association de degré supérieur ne comporte pas d'entité d'association, une table fille est tout de même dérivée de l'association, sans attribut propre. La table porte le nom de l'association. Sa clé primaire et les multiplicités des associations avec les tables mères sont déterminés selon les règles qui s'appliquent à une table fille dérivée d'une entité d'association présente sur une association de degré supérieur.

Cependant, une différence importante est à prendre en considération. Si les multiplicités maximales sont à *plusieurs* sur tous les arcs de l'association, la table fille pourrait avoir une clé primaire composée qui n'est pas valide. Il s'agit pour le modélisateur d'analyser le contexte et de tenter d'établir si elle est acceptable. En cas de doute, il aura à appliquer l'astuce vue plus tôt concernant les associations binaires *plusieurs à plusieurs* sans entité d'association et consistant à créer une clé primaire simple et artificielle avec la mention {PK auto}, tout en conservant dans la table fille les clés primaires des tables mères mais uniquement à titre de clés étrangères avec mention {FK}{Cascade}.

La figure 2-28 montre un modèle conceptuel avec association de degré supérieur sans entité d'association et où on retrouve au moins une multiplicité maximale 1. On peut donc effectuer une simplification de la clé primaire. La table fille nommée **Comporte** aura une clé primaire combinant les clés primaires des autres tables **Réservation** et **Vol**.

Dans le modèle relationnel dérivé, illustré à la figure 2-29, on notera que la clé primaire de la table mère **Tarif**, **Code de tarif**, est tout de même copiée dans la table fille **Comporte**. Bien que clé étrangère, elle ne constitue

cependant pas un élément de la clé primaire composée de cette dernière car la multiplicité maximale de son côté est 1. De plus la mention {Cascade} N'EST PAS PRÉSENTE, une conséquence de la simplification.

Puisque qu'une multiplicité maximale de 1 est présente sur un des arcs de l'association, il n'y a pas lieu de créer une clé primaire artificielle. La combinaison des attributs **No réservation** et **No vol** assure l'unicité d'accès à une ligne de la table.

FIGURE 2-28 **Modèle conceptuel avec une association de degré supérieur**

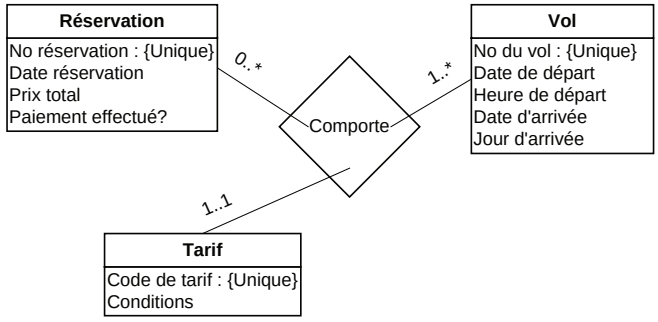
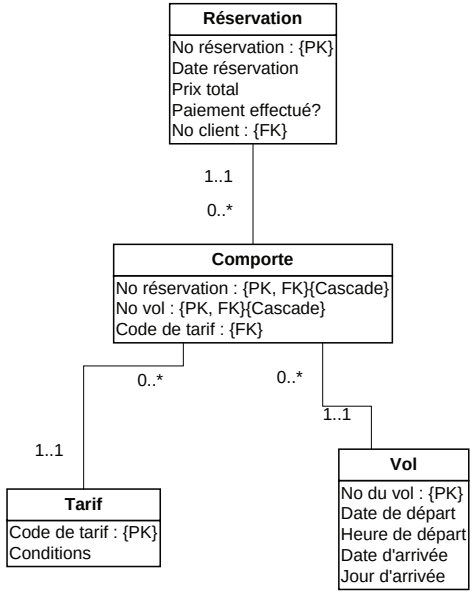


FIGURE 2-29 **Modèle relationnel dérivé du modèle 2-28**



Il y a lieu de rappeler que toutes les associations de la table fille sont 1..1 côté mère et 0..* côté fille. Il s'agit d'une règle incontournable : toute association de degré supérieur, qu'elle porte une entité d'association ou non, donne lieu à une table fille dont les associations portent des multiplicités 1..1 côté mère et 0..* côté fille.

Si les multiplicités maximales avaient été *plusieurs* sur tous les arcs du modèle conceptuel, la table **Comporte** exigerait probablement une clé primaire simple et artificielle, **No comporte** {PK auto}, avec les attributs **No réservation**, **No vol** et **Code tarif** portant tous la mention {FK}{Cascade}. C'est la responsabilité du modélisateur d'établir la validité de la clé primaire composée et de la remplacer le cas échéant par une clé simple.

Ces deux illustrations portant sur la conversion d'une association de degré supérieur montre la relative complexité des règles de dérivation des tables et des associations qui s'appliquent dans un tel cas. Nous ne saurons trop insister sur l'importance de décomposer une association de degré supérieur qui comporte une multiplicité maximale *un* ou qui montre une dépendance fonctionnelle entre deux entités associées, pour réduire au minimum le nombre d'associations de degré supérieur et faire en sorte que celles qui ne peuvent être décomposées ne portent qu'une multiplicité maximale *plusieurs*.

Éliminer les multiplicités maximales valant 1 par décomposition des associations de degré supérieur : Avant d'aborder la conversion d'un modèle conceptuel en son pendant relationnel, il faut s'assurer de la décomposition des associations de degré supérieur portant des multiplicités maximales de 1.

L'association d'héritage

Les tables dérivées d'une association ou de plusieurs associations d'héritage rattachées au même supertype seront de natures fort variées selon qu'il y a présence ou non de contraintes inter-associations dans le modèle conceptuel.

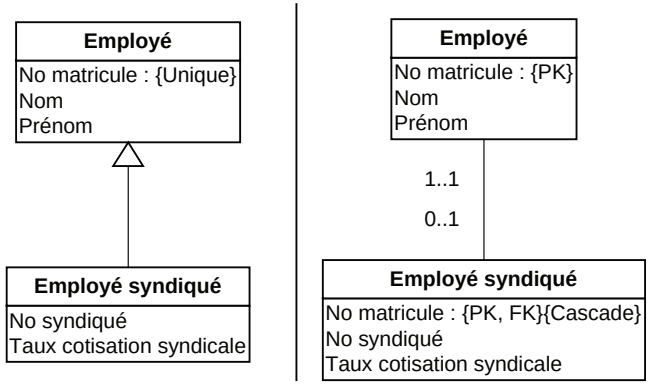
Le tableau 2-2 résume les quatre situations possibles et la nature des tables dérivées.

Nous allons illustrer à tour de rôle chaque situation à l'aide d'un exemple. Toutes les figures montrent à gauche un modèle conceptuel avec une ou des associations d'héritage et à droite le modèle relationnel résultant.

TABLEAU 2-2 **Tables dérivées selon la nature des contraintes sur les associations d'héritage**

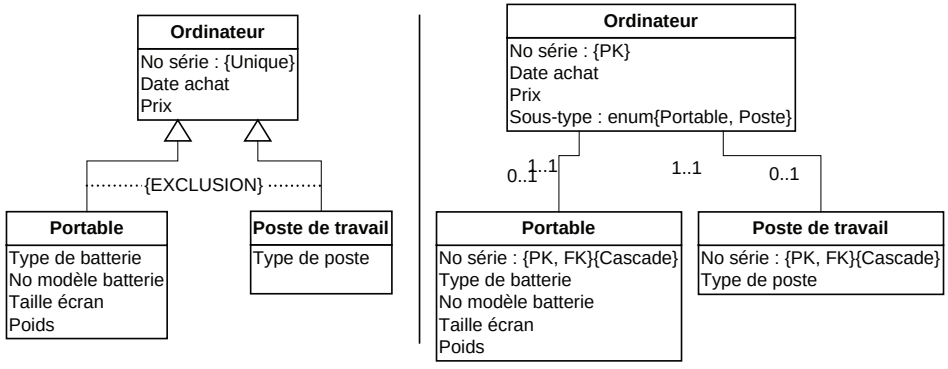
Contrainte	Tables dérivées	Clé primaire	Clé étrangère
Aucune	Une table *mère* pour le supertype, une table *fille* pour chaque sous-type.	Identifiant du supertype pour chaque table.	La clé primaire des tables des sous-types est aussi étrangère avec intégrité référentielle (ajout, suppression, mise à jour). Multiplicités 1..1 côté mère, 0..1 côté fille.
Exclusion	Une table *mère* pour le supertype, une table *fille* pour chaque sous-type. Un attribut présent dans la table mère spécifie le sous-type d'une ligne de la table le cas échéant.	Identifiant du supertype pour chaque table.	La clé primaire des tables des sous-types est aussi étrangère avec intégrité référentielle (ajout, suppression, mise à jour). Multiplicités 1..1 côté mère, 0..1 côté fille.
Totalité	Une seule table portant le nom du supertype et regroupant les attributs du supertype et de tous les sous-types. Un attribut booléen pour chaque sous-type permettant d'établir à quels sous-types appartient une ligne.	Identifiant du supertype.	N/A
Partition	Une seule table portant le nom du supertype et regroupant les attributs du supertype et de tous les sous-types. Un attribut **non nul** présent dans la table spécifie le seul sous-type admissible pour une ligne.	Identifiant du supertype.	N/A

FIGURE 2-30 **Cas 1 : aucune contrainte**



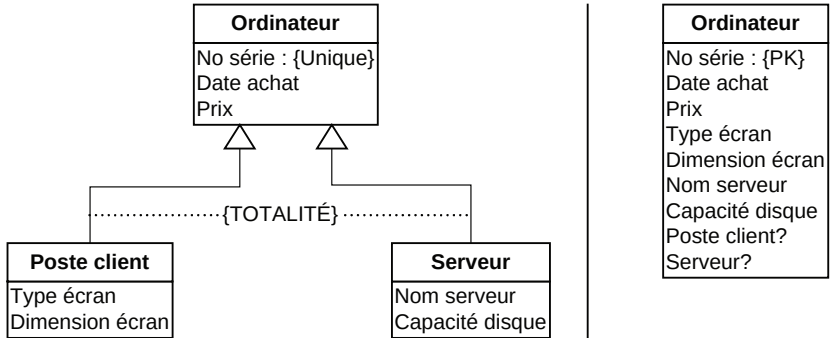
Le modèle 2-30 ne montre qu'un sous-type. Si l'entité **Employé** devait avoir plus d'un sous-type, chaque sous-type donnerait lieu à une table avec la clé primaire **No matricule**.

FIGURE 2-31 **Cas 2 : Contrainte d'exclusion**



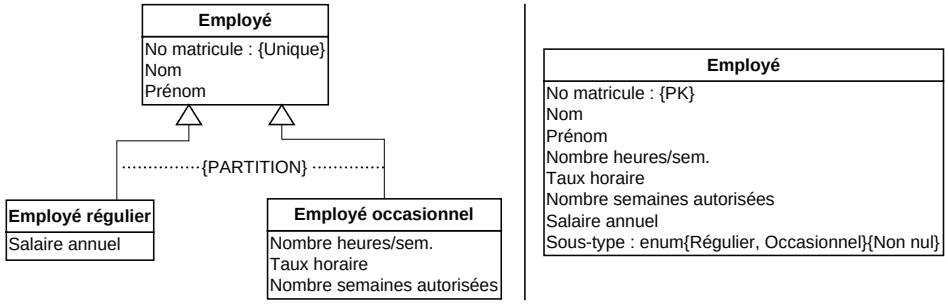
Puisqu'une exclusion impose au plus un seul sous-type à une occurrence du supertype, l'attribut **Sous-type** permet de spécifier le sous-type de l'occurrence. Cet attribut peut être nul car l'occurrence peut n'appartenir à aucun sous-type.

FIGURE 2-32 **Cas 3 : Contrainte de totalité**



Une contrainte de totalité spécifie qu'une occurrence du supertype peut avoir les attributs de tous ses sous-types et donc être de tous les sous-types à la fois. Deux attributs booléens, **Poste client ?** et **Serveur ?**, permettent d'établir les sous-types auxquels une ligne de la table dérivée appartient.

FIGURE 2-33 **Cas 4 : Contrainte de partition**



Une contrainte de partition stipule qu'une occurrence du supertype doit appartenir obligatoirement à un des sous-types. D'où la présence d'un attribut discriminant obligatoire, **Sous-type**, qui rattache l'occurrence à un des deux sous-types admissibles.

Les contraintes inter-associations

Seules les contraintes inter-associations sur l'héritage ont un impact sur les tables dérivées du modèle conceptuel comme nous l'avons vu à la rubrique précédente. Les contraintes inter-associations sur les associations binaires ne sont prises en compte que pour la réalisation du modèle physique qui découle du modèle relationnel. De manière à simplifier le modèle relationnel, les contraintes inter-associations sur les associations binaires ne sont pas réintroduites dans le modèle relationnel. Nous aurons le loisir d'étudier au chapitre 3 l'impact des contraintes inter-associations sur le modèle physique de données.

CAS DE MODÉLISATION LOGIQUE DES DONNÉES

Nous débutons cette section en proposant un tableau de synthèse des règles de dérivation. Ce tableau sera utilisé dans les études de cas pour établir une démarche systématique de dérivation des tables et des associations menant à un modèle relationnel complet et valide.

TABLEAU 2-3 **Tableau de synthèse sur les règles de dérivation du modèle logique**

Règle et objet de la règle	Tables dérivées	Association entre les tables	Intégrité référentielle
2-1 Entité forte	Une table avec l'identifiant de l'entité comme clé primaire.	N/A	N/A
2-2 Entité d'association	Une table fille pour l'entité d'association. Une table mère pour chacune des entités associées. La clé primaire de la fille est la combinaison des clés primaires des tables mères. Cette clé primaire peut être simplifiée une seule fois si un arc comporte une multiplicité maximale de 1.	*Pour une association binaire : Deux* associations comportant toutes des multiplicités 1..1 du côté de chaque table mère ; du côté de la table fille les multiplicités sont reprises du modèle conceptuel. Il s'agit dans chaque cas des multiplicités présentes à la terminaison d'arrivée de l'association au modèle conceptuel. *Pour une association de degré supérieur : N* associations comportant toutes des multiplicités 1..1 du côté d'une table mère et 0..* côté de la table fille.	Toutes les clés primaires des tables mères copiées dans la table fille sont des clés étrangères avec intégrité référentielle en ajout, suppression et mise à jour (mention {Cascade}), sauf dans le cas d'une simplification où la clé étrangère retirée de la clé de la table fille assure l'intégrité référentielle en ajout seulement (PAS de mention {Cascade}).
2-3 Association de composition	Une table mère pour le composite et une table fille pour le composant. La clé primaire de la table fille combine son identifiant et la clé primaire de la mère.	L'association porte les multiplicités 1..1 du côté de la table mère et du côté fille, les multiplicités présentes au modèle conceptuel du côté de l'entité composant.	La portion de la clé primaire de la table fille provenant de la table mère est une clé étrangère avec intégrité référentielle en ajout, suppression et mise à jour (mention {Cascade}).
2-4 Association d'héritage	La nature de la ou des tables dérivées dépend de la présence d'une contrainte inter-association. (voir tableau 2-2) Dans certains cas, un ou des attributs de la seule table dérivée servent à établir le ou les sous-types d'un tuple.	En absence de contrainte inter-association ou en présence de contrainte d'*exclusion*, toutes les associations portant des multiplicités 0..1-1..1, 1..1 sont du côté de la table mère. Pour les autres contraintes, aucune association est présente puisqu'une seule table est dérivée.	En absence de contrainte inter-association ou en présence de contrainte d'*exclusion*, la clé primaire de la table fille provenant de la table mère est une clé étrangère avec intégrité référentielle en ajout, suppression et mise à jour (mention {Cascade}).
2-5 Association binaire, cas 1-1	La table retenue comme table fille sera celle dérivée de la seule entité marquée d'une multiplicité minimale 0, ou la seule comportant une autre association. Si aucune des deux entités ne rencontre seule une de ces deux conditions, la table mère est choisie arbitrairement.	Multiplicités reprises du modèle conceptuel.	La clé primaire de la table mère est une clé étrangère dans la table fille avec intégrité référentielle appliquée en ajout seulement (PAS de {Cascade}.

TABLEAU 2-3 **Tableau de synthèse sur les règles de dérivation du modèle logique (*suite*)**

Règle et objet de la règle	Tables dérivées	Association entre les tables	Intégrité référentielle
2-6 Association binaire, cas 1-*	La table retenue comme table fille est celle dérivée de l'entité côté *plusieurs*.	Multiplicités reprises du modèle conceptuel.	La clé primaire de la table mère est une clé étrangère dans la table fille avec intégrité référentielle appliquée en ajout seulement (PAS de mention {Cascade}.
2-7 Association binaire, cas *-*	Une table fille ne comportant aucun attribut propre est dérivée de l'association et elle porte son nom. Une table mère pour chacune des *deux* entités associées. La clé primaire de la fille est la combinaison des *deux* clés primaires des tables mères. Elle pourrait selon le contexte ne pas être valide. Le cas échéant une clé primaire simple et artificielle avec la mention {PK auto} est substituée à la clé primaire composée.	Deux associations comportant toutes des multiplicités 1..1 du côté de la table mère ; du côté de la table fille les multiplicités sont reprises du modèle conceptuel. Il s'agit dans chaque cas des multiplicités présentes à la terminaison d'arrivée de l'association au modèle conceptuel.	Les clés primaires des tables mères copiées dans la table fille sont des clés étrangères avec intégrité référentielle en ajout, suppression et mise à jour (mention {Cascade})
2-8 Association réflexive	Si elle comporte une entité d'association, la règle 2-2 donnée plus haut s'applique. Sinon les règles d'une association binaire 1-1, 1-*, *-* s'appliquent selon le cas.		
2-9 Association de degré supérieur	Une table fille est créée pour l'association. Si l'association est dotée d'une entité d'association, ses attributs sont ajoutés à la table fille. Sinon la table fille ne sera dotée d'aucun attribut propre et elle portera le nom de l'association. Une table mère est dérivée pour chacune des *n* entités associées. La clé primaire de la fille est la combinaison des clés primaires des tables mères et si une multiplicité maximale à *un* est présente sur un des arcs, la clé primaire peut être simplifiée qu'une seule fois. Si les multiplicités maximales sont toutes à *plusieurs* et que l'association ne dispose pas d'entité d'association, la clé primaire composée pourrait selon le contexte ne pas être valide. Le cas échéant une clé primaire simple et artificielle avec la mention {PK auto} est substituée à la clé primaire composée.	*N* associations comportant toutes des multiplicités 1..1 du côté d'une table mère et 0..* du côté de la table fille.	Les clés primaires des tables mères copiées dans la table fille sont des clés étrangères avec intégrité référentielle en ajout, suppression et mise à jour (mention {Cascade}), sauf dans le cas d'une simplification où la clé étrangère retirée de la clé de la table fille assure l'intégrité référentielle en ajout seulement (PAS de mention {Cascade})

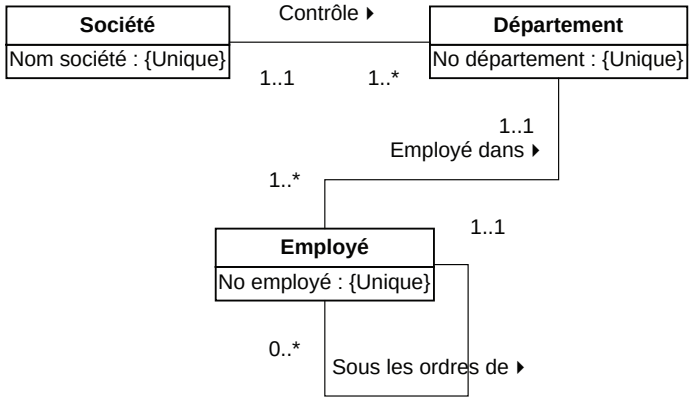
La démarche préconisée pour assurer le passage du modèle conceptuel au modèle relationnel reprend les éléments du tableau 2-3 :

1. Appliquer systématiquement la règle de modélisation conceptuelle 1-5 visant à décomposer les associations de degré supérieur s'il y a lieu ;
2. Convertir les associations d'héritage ;
3. Convertir les entités fortes qui restent ;
4. Convertir les associations de composition ;
5. Convertir les entités d'association tant sur les associations binaires que sur celles de degré supérieur ;
6. Convertir les associations binaires restantes et les associations réflexives ;
7. Convertir les associations de degré supérieur restantes.

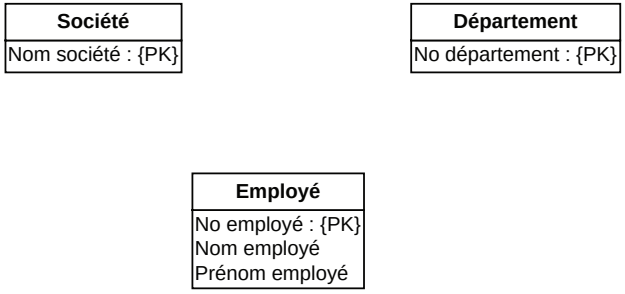
Les cas qui suivent sont tous tirés de ceux discutés au chapitre 1. On y explique de manière détaillée l'application des règles de dérivation du modèle relationnel à partir d'un modèle conceptuel introduit au chapitre précédent.

CAS 2-1 SOCIÉTÉ ET DÉPARTEMENT

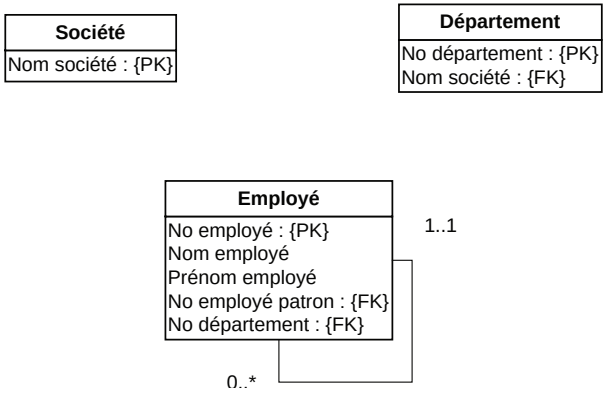
Traduire ce modèle conceptuel en son pendant relationnel tout en appliquant les règles de dérivation appropriées.



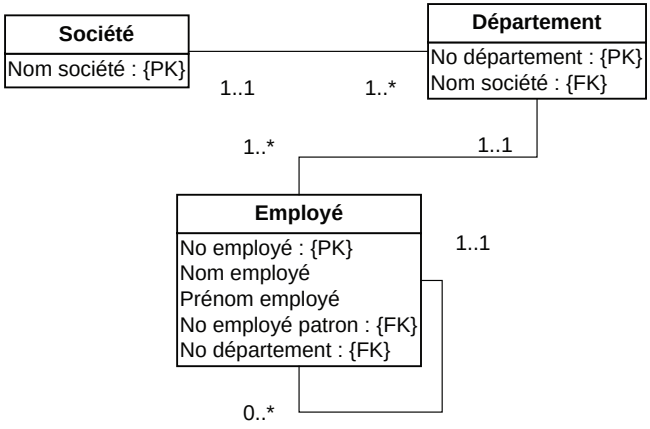
Le modèle ne comporte que des entités fortes dont on dérive trois tables.



Une association binaire réflexive UN À PLUSIEURS donne lieu à une table associée à elle-même.



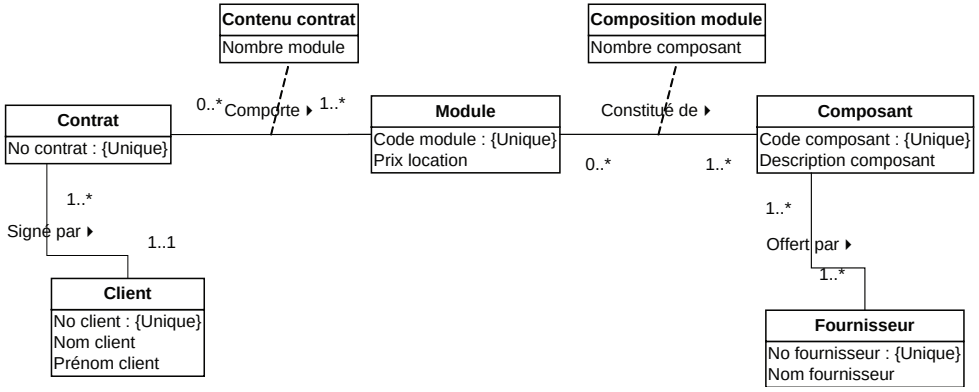
L'association **Contrôle** et l'association **Employé dans** sont toutes deux de type BINAIRE UN À PLUSIEURS. La table **Département** est à la fois fille pour l'association **Contrôle** et mère pour l'association **Employé dans**.



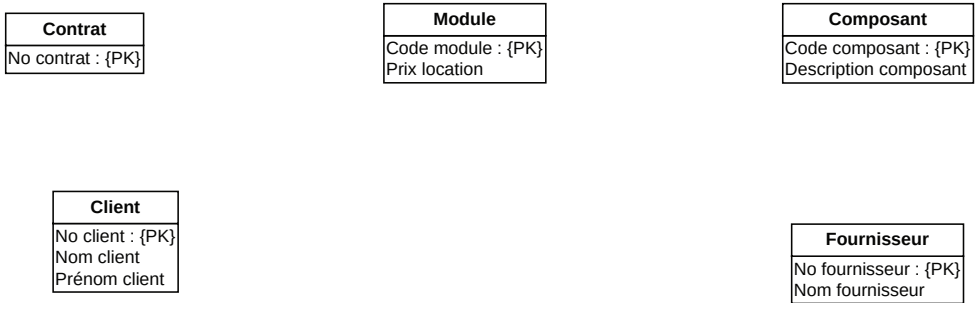
Cela explique que sa clé primaire **No département** est reproduite comme clé étrangère dans la table **Employé** et qu'elle possède une clé étrangère, **Nom société**, soit la clé primaire de sa table mère **Société**.

CAS 2-2 LOCATEUR D'ABRIS

Traduire ce modèle conceptuel en son pendant relationnel tout en appliquant les règles de dérivation appropriées.

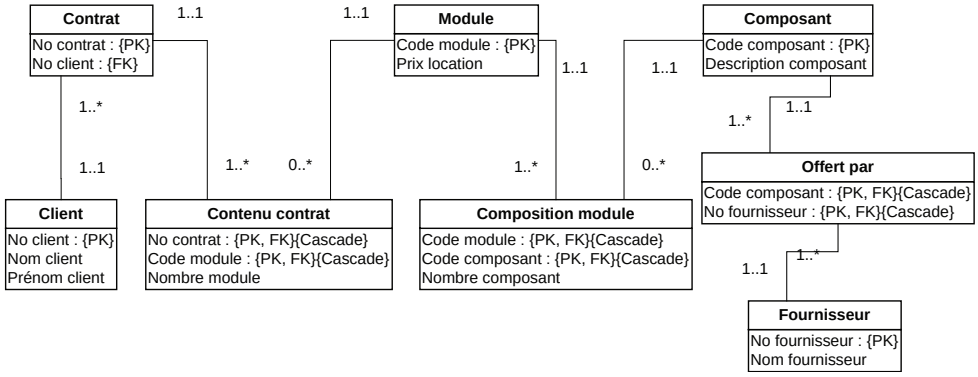


Le modèle comporte cinq entités fortes et deux faibles, les entités d'association **Contenu contrat** et **Composition module**. Cinq tables sont dérivées des entités fortes.



Les entités d'association, des entités faibles, donnent lieu à deux tables filles, **Contenu contrat** et **Composition module**, comportant une clé primaire composée de celles de leurs mères dotées des contraintes d'intégrité appropriées. Les multiplicités doivent être 1..1 systématiquement du côté des tables mères. Les multiplicités du côté de la table fille sont déterminées par les multiplicités présentes sur les terminaisons d'arrivée dans l'association au modèle conceptuel.

Quant à l'entité d'association sur **Comporte**, si la lecture de l'association est faite de **Contrat** vers **Module**, la terminaison d'arrivée montre des multiplicités 1..*. La multiplicité 1..* sera donc présente du côté de la table fille sur l'association qui la relie à la table **Contrat**. Si la lecture de l'association est faite de **Module** vers **Contrat**, la terminaison d'arrivée montre des multiplicités 0..*. La multiplicité 0..* sera donc présente du côté de la table fille sur l'association qui la relie à la table **Module**.



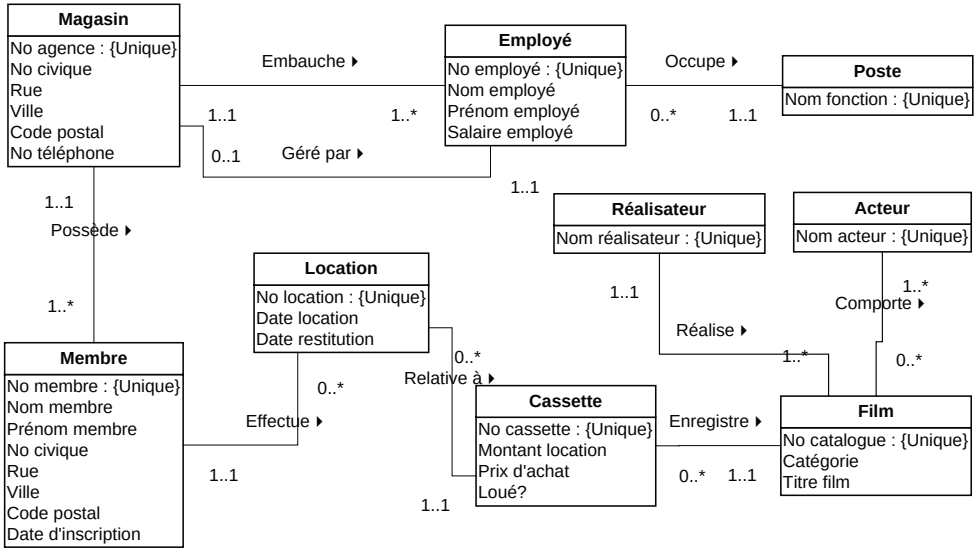
On effectue la même réflexion pour l'entité d'association sur **Constitué de**. Si la lecture de l'association est faite de **Module** vers **Composant**, la terminaison d'arrivée montre des multiplicités 1..*. La multiplicité 1..* sera donc présente du côté de la table fille sur l'association qui la relie à la table **Module**. Si la lecture de l'association est faite de **Composant** vers **Module**, la terminaison d'arrivée montre des multiplicités 0..*. La multiplicité 0..* sera donc présente du côté de la table fille sur l'association qui la relie à la table **Composant**.

L'association binaire UN À PLUSIEURS **Signé par** établit que **Client** doit agir comme table mère, d'où la présence de sa clé dans la table **Contrat** comme clé étrangère.

L'association binaire **Offert par**, de type PLUSIEURS À PLUSIEURS, exige une table fille combinant les clés primaires des tables **Composant** et **Fournisseur**. La multiplicité est 1..* pour les deux associations du côté de la table fille. La clé primaire composée de cette table est valide ; elle ne peut avoir de doublon.

CAS 2-3 VIDÉO CLUB

Traduire ce modèle conceptuel en son pendant relationnel tout en appliquant les règles de dérivation appropriées.



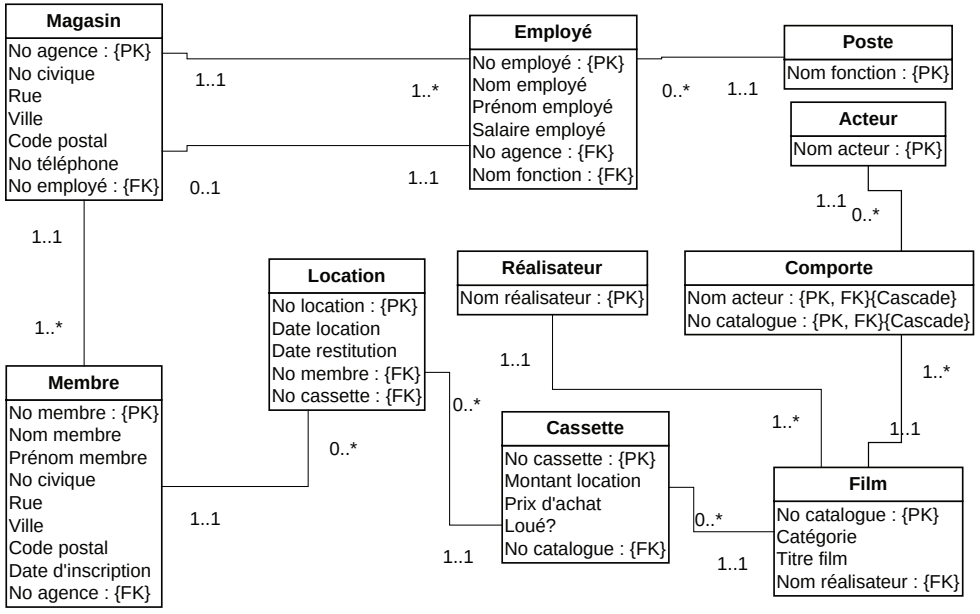
Toutes les entités sont fortes et les associations binaires sont toutes de type UN À PLUSIEURS, sauf deux exceptions, **Géré par**, qui est de type UN À UN et **Comporte** de type PLUSIEURS À PLUSIEURS.

Dans le cas de **Géré par**, la multiplicité minimale est de 0 du côté de **Magasin**, la table dérivée **Magasin** devient donc la table fille pour cette association.

Quant à l'association **Comporte** de type PLUSIEURS À PLUSIEURS, elle donne lieu à une table fille portant le nom de l'association et dupliquant les clés primaires des tables **Film** et **Acteur** à titre de clés étrangères. Le modèle conceptuel montre que la destination d'arrivée de **Comporte** à partir de **Acteur** porte 0..*. On retrouvera donc 0..* du côté de la table fille pour l'association la liant à la table **Acteur**. Pour la lecture à partir de l'entité **Film**, la multiplicité 1..* est présente. On retrouvera donc 1..* du côté de la table fille pour l'association la liant à la table **Film**.

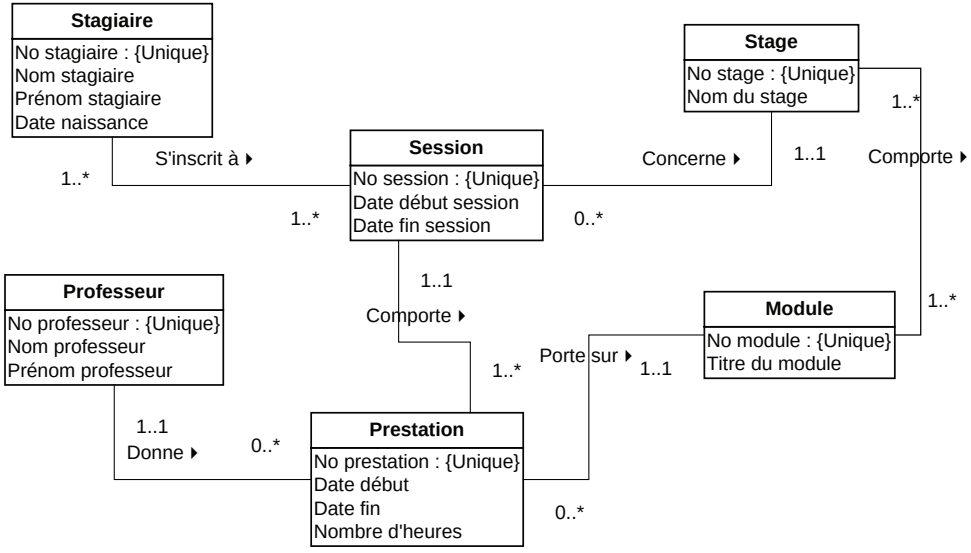
Elle possède une clé primaire composée valide : un acteur ne peut paraître deux fois au générique du même film. L'intégrité référentielle s'applique aux deux clés étrangères en ajout, en suppression et en mise à jour (mention {Cascade}).

Pour toutes les autres associations qui sont de type un à plusieurs, les tables dérivées du côté PLUSIEURS sont des tables filles dupliquant à titre de clé étrangère la clé primaire de leur table mère.



CAS 2-4 GESTION DE STAGE

Traduire ce modèle conceptuel en son pendant relationnel tout en appliquant les règles de dérivation appropriées.

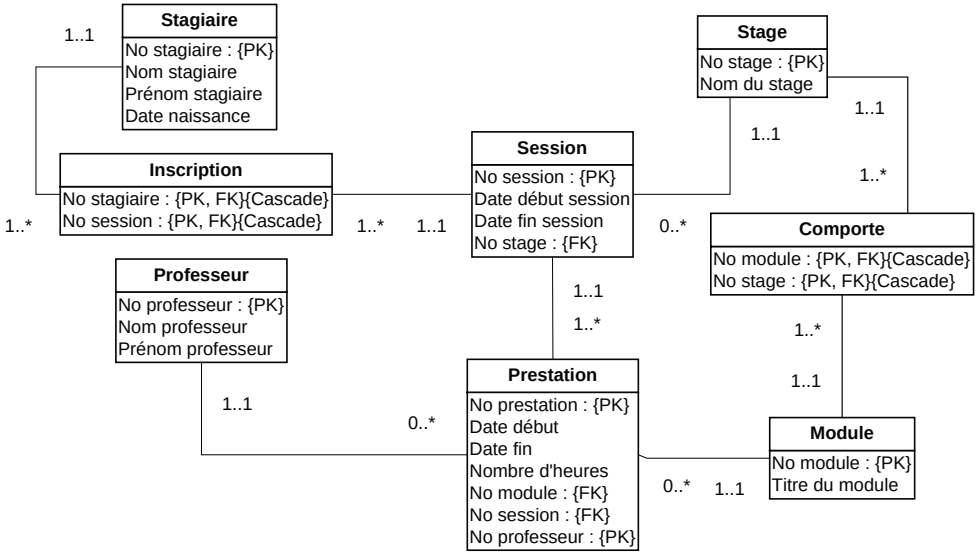


On ne retrouve dans ce modèle conceptuel que des entités fortes, dont six tables sont dérivées. On note par ailleurs la présence de deux associations binaires PLUSIEURS À PLUSIEURS, **S'inscrit à** et **Comporte**, qui donnent lieu à deux tables filles. La table dérivée de l'association **S'inscrit à** a pour nom **Inscription**, ce qui semble dans le contexte un nom beaucoup plus approprié que **S'inscrit à**. Celle dérivée de **Comporte** porte le nom de l'association, une convention acceptable dans ce cas. Le nombre total de tables est donc de huit.

Les tables dérivées des associations PLUSIEURS À PLUSIEURS ont une clé primaire composée des clés primaires des tables mères. Leur clé primaire est valide. Chaque élément de la clé composée est aussi une clé étrangère sur laquelle l'intégrité référentielle s'applique en ajout, en suppression et en mise à jour (mention {Cascade}).

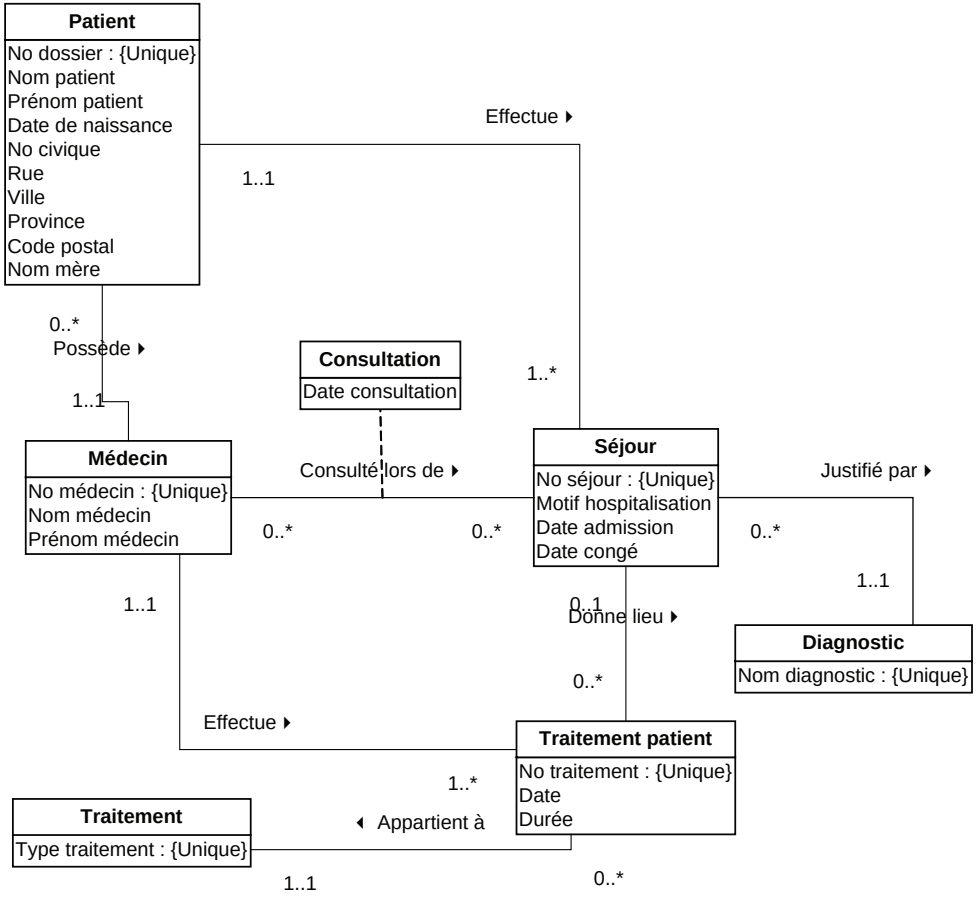
La table **Prestation** comporte trois clés étrangères car l'entité dont elle dérive possède trois associations UN À PLUSIEURS et les multiplicités maximales du côté **Prestation** sont toutes PLUSIEURS(*). **Prestation** agit donc comme table fille pour les trois associations.

La table **Session** est table fille pour l'association **Concerne**.



CAS 2-5 DOSSIER PATIENT

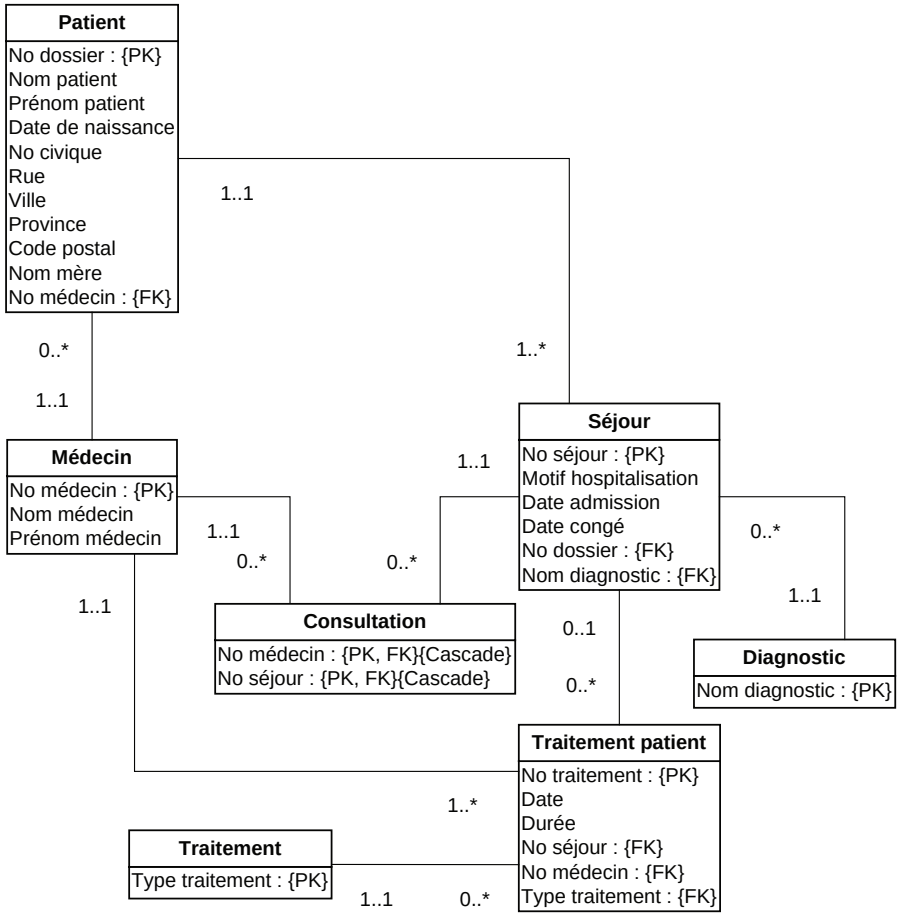
Traduire ce modèle conceptuel en son pendant relationnel tout en appliquant les règles de dérivation appropriées.



Ce modèle comporte six entités fortes et une entité faible : **Consultation**. Le modèle relationnel dérivé comporte donc sept tables.

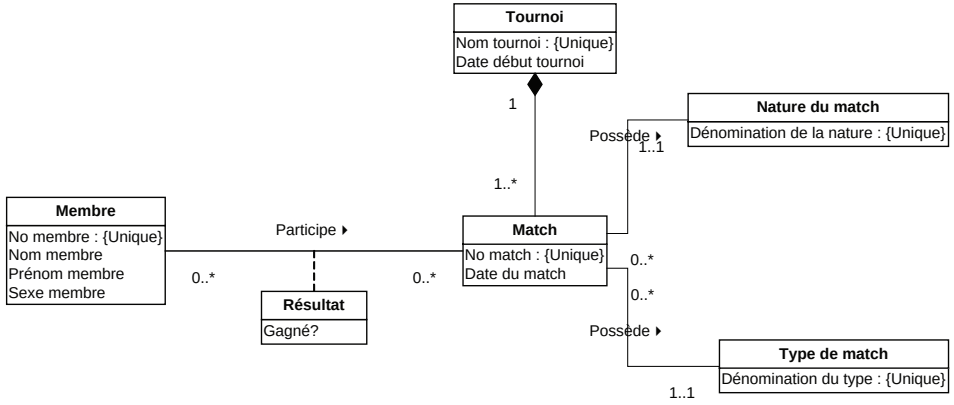
La table **Consultation** est une table fille associée aux tables **Médecin** et **Séjour**. Sa clé primaire est composée des clés primaires des deux tables mères. Chaque élément de la clé composée est aussi clé étrangère sur laquelle l'intégrité référentielle s'applique en ajout, en suppression et en mise à jour (mention {Cascade}).

À l'exception de l'association **Consulté lors de** qui comporte une entité faible dont la table fille **Consultation** est dérivée, toutes les associations sont de type binaire UNE À PLUSIEURS. Les tables filles sont du côté PLUSIEURS pour ce type d'association. Il s'agit des tables **Patient**, **Séjour**, **Traitement patient**. Cette dernière dérive d'une entité comportant trois associations avec des multiplicités maximales plusieurs de son côté. Elle possède en conséquence trois clés étrangères.



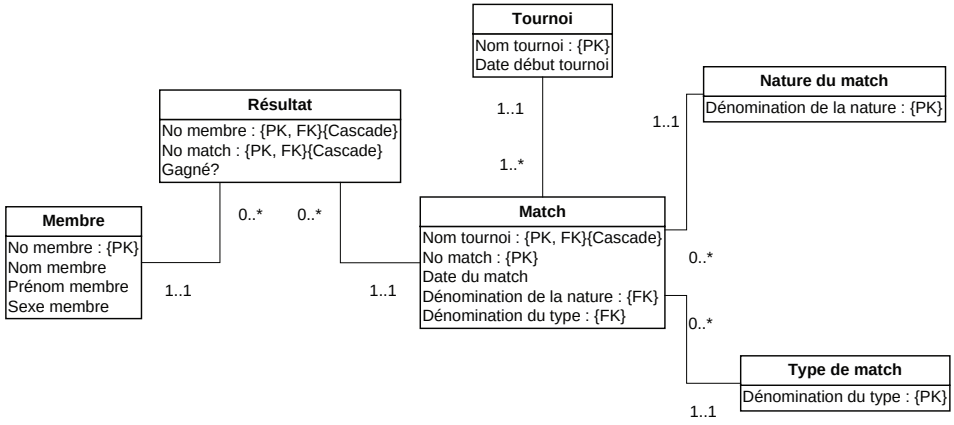
CAS 2-6 MEMBRE CLUB DE TENNIS

Traduire ce modèle conceptuel en son pendant relationnel tout en appliquant les règles de dérivation appropriées.



Le modèle comporte quatre entités fortes et deux entités faibles. **Résultat** est une entité d'association et **Match**, une entité composant.

Match donne lieu à une table dont la clé primaire combine son identifiant, **No match**, et la clé primaire de son composite **Tournoi**: **Nom tournoi**. **Nom tournoi** est aussi une clé étrangère sur laquelle s'applique l'intégrité référentielle en ajout, en suppression et en mise à jour (mention {Cascade}).

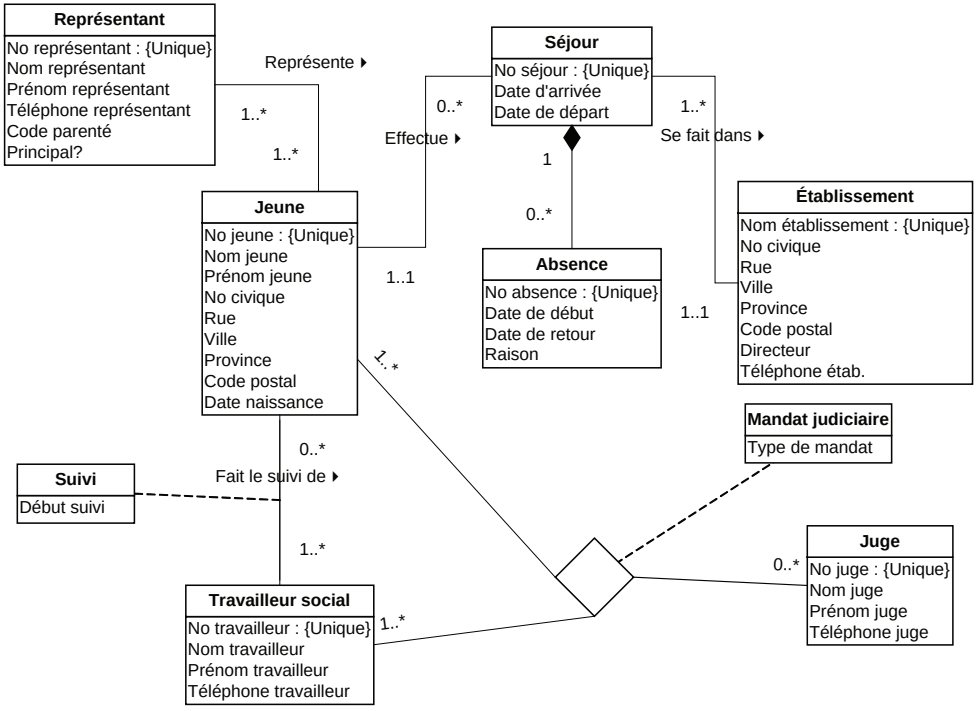


Résultat donne lieu à une table fille créée selon les règles habituelles de dérivation pour une entité d'association.

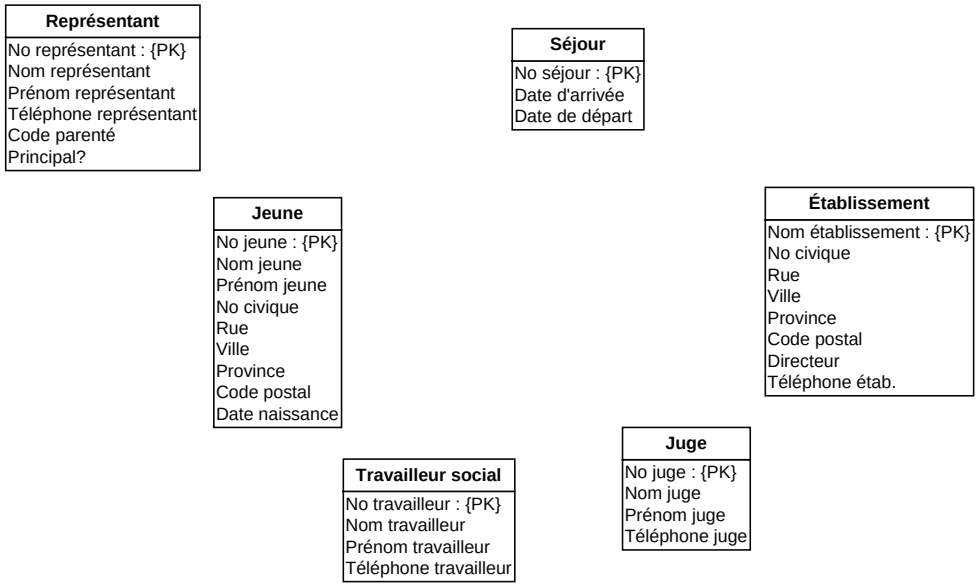
Puisque l'entité **Match** comporte de plus deux associations binaires de type UN À PLUSIEURS appelées toutes les deux **Possède**, deux clés étrangères doivent être présentes dans cette table fille pour assurer les associations.

CAS 2-7 SÉJOUR DANS UN ÉTABLISSEMENT JEUNESSE

Traduire ce modèle conceptuel en son pendant relationnel tout en appliquant les règles de dérivation appropriées.



Le modèle comporte six entités fortes et trois entités faibles. Aucune association d'héritage n'est présente. Les tables dérivées des entités fortes sont montrées ci-après. Quant aux entités faibles, **Suivi** et **Mandat judiciaire**, elles sont des entités d'association et **Absence** est une entité composant.



L'association de composition est d'abord considérée. L'association de composition donne lieu à une table **Absence** associée à la table **Séjour** dont la clé primaire est reproduite dans **Absence**. **No Séjour** s'ajoute donc à l'attribut **No absence** pour constituer la clé primaire composée de **Absence**. La multiplicité 0..* du côté du composant est reprise du modèle conceptuel.

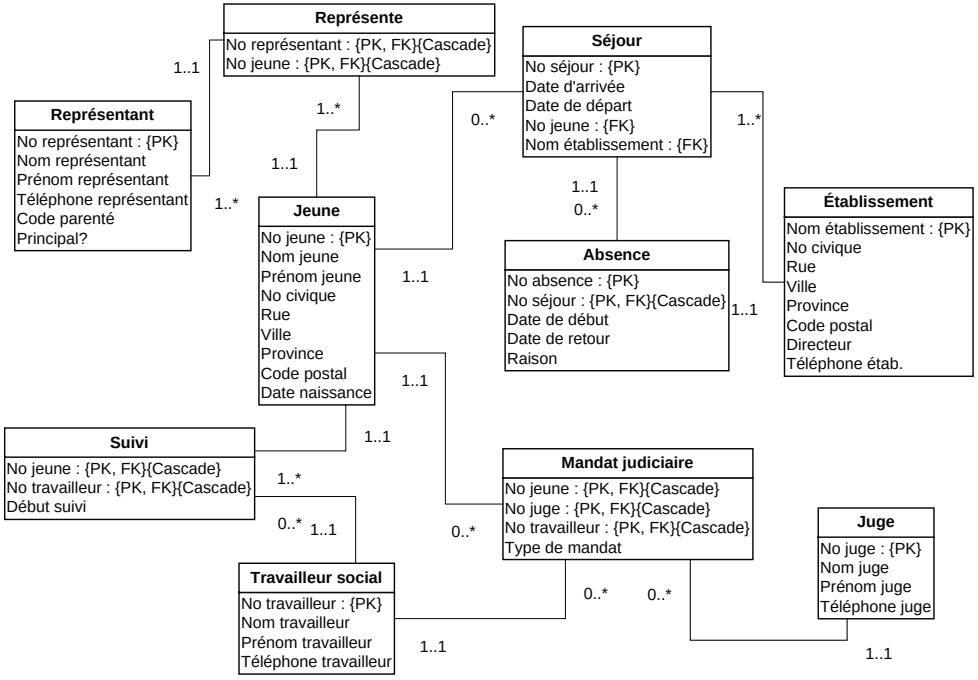
Les entités d'association seront traitées ensuite. L'entité d'association **Suivi** et l'association de degré supérieur **Mandat judiciaire** ne comportent que des multiplicités maximales plusieurs et de plus **Mandat judiciaire** n'est pas décomposable. La règle 2-2 sera donc appliquée sans possibilité de simplification de la clé primaire composée dans chaque cas.

Des tables seront dérivées de **Suivi** et **Mandat judiciaire** comportant une clé primaire composée des clés primaires des tables dérivées des entités associées. Leur attribut propre est présent dans la table.

La multiplicité du côté de la table **Suivi** est reprise du modèle conceptuel. Selon la convention habituelle, au modèle conceptuel la lecture de l'association de **Jeune** vers **Travailleur social** montre la multiplicité 1..* sur la terminaison d'arrivée. La multiplicité du côté de la table fille pour l'association la liant à la table **Jeune** sera donc 1..*. La lecture dans l'autre sens montre 0..*. La multiplicité du côté de la table fille pour l'association la liant à la table **Travailleur social** sera donc 0..*. Du côté des tables mères, les multiplicités sont toutes 1..1.

Mise en garde : les multiplicités du côté de la table **Mandat judiciaire** NE SONT PAS REPRISES du modèle conceptuel. Elles doivent toutes être 0..* car il s'agit d'une association de degré supérieur. Du côté des tables mères cependant, elles sont toutes 1..1.

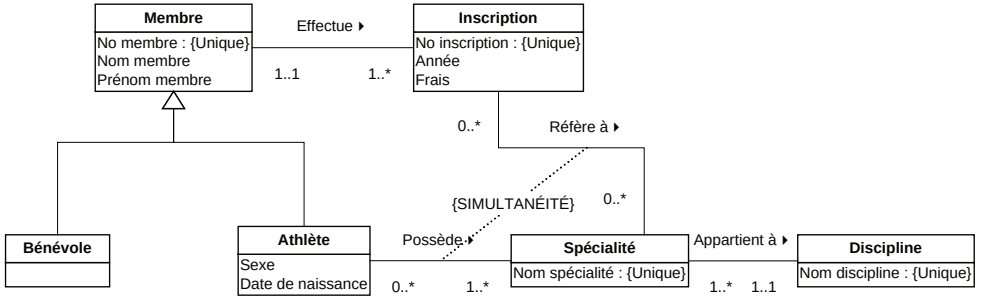
L'association binaire **Représente** est du type PLUSIEURS À PLUSIEURS. Une table en est dérivée dont la clé composée est valide. Les autres associations binaires sont du type UN À PLUSIEURS, ce sont **Effectue** et **Se fait dans**. Elles génèrent deux clés étrangères dans la table fille **Séjour**.



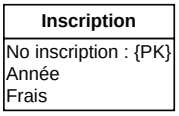
Ce cas démontre certaines limites quant à l'expression dans le modèle relationnel des contraintes de multiplicité d'une association de degré supérieur non décomposable. En effet les multiplicités minimales du côté de la table fille seront systématiquement 0 et n'ont rien à voir avec celles du modèle conceptuel. La raison en est que la sémantique des multiplicités d'une association de degré supérieure en UML n'a pas son équivalent dans le modèle relationnel. Ce n'est pas le cas des associations binaires, réflexives ou non, où les multiplicités peuvent être reprises telles quelles dans le modèle relationnel.

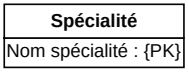
CAS 2-8 CLUB OLYMPIQUE

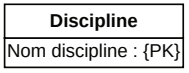
Traduire ce modèle conceptuel en son pendant relationnel tout en appliquant les règles de dérivation appropriées.



Trois entités fortes sont présentes, si on exclut **Membre** liée par une association d'héritage : **Inscription**, **Spécialité** et **Discipline**. On en dérive trois tables.

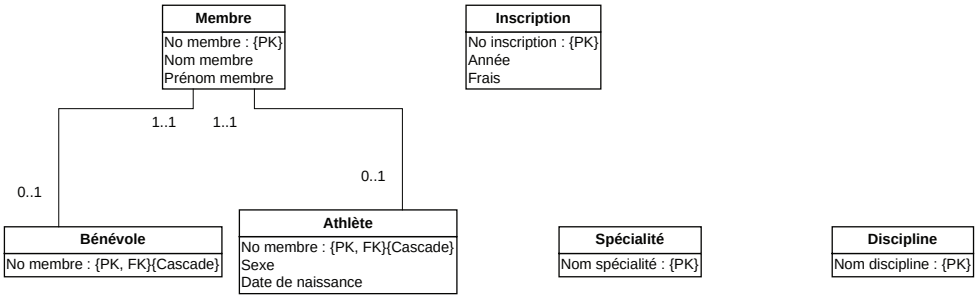






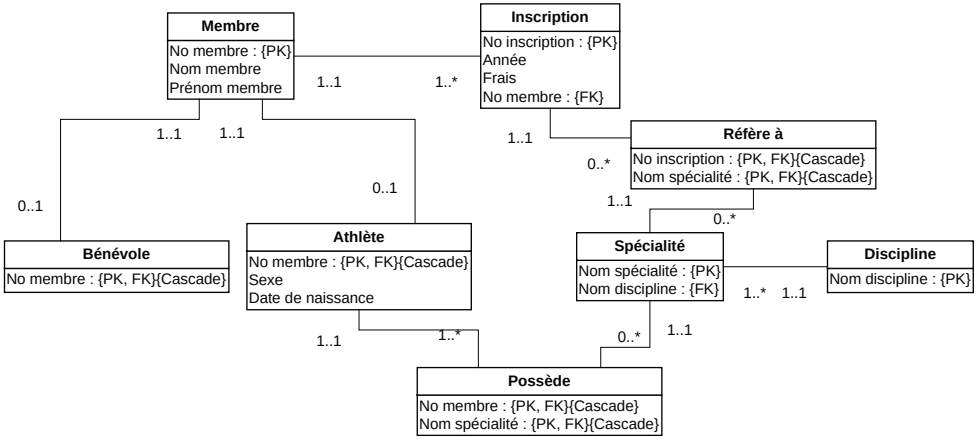
Les associations d'héritage sont considérées en premier lieu. Aucune contrainte inter-association n'est présente sur les associations d'héritage. En vertu du tableau 2-2, la première règle portant sur les associations d'héritage s'applique : une table pour chaque entité. La clé primaire des tables dérivées des sous-types possède comme clé primaire, une copie de la clé primaire du supertype, soit **No membre**. Cette copie est aussi clé étrangère avec intégrité référentielle en ajout, suppression et mise à jour.

Les associations liant les tables dérivées de l'association d'héritage portent systématiquement les multiplicités 1..1 et 0..1. La multiplicité 1..1 est du côté de la table mère **Membre**.



Les associations binaires peuvent maintenant être prises en considération. **Possède** et **Réfère** sont du type PLUSIEURS À PLUSIEURS et vont donner lieu à deux nouvelles tables.

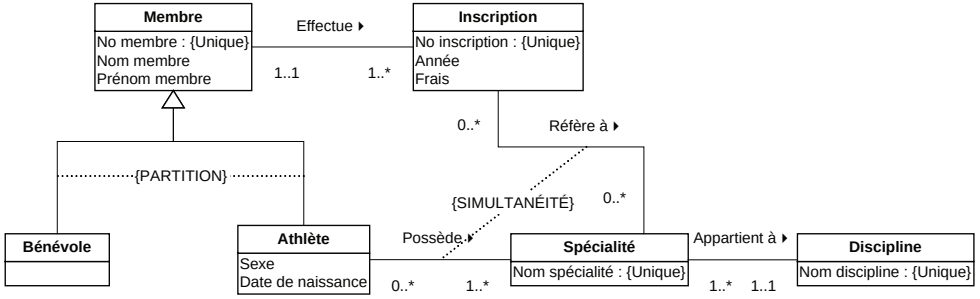
Quant à **Appartient à**, de type UN À PLUSIEURS, elle nécessite la copie de **Nom discipline** dans la table fille **Spécialité** à titre de clé étrangère. Il en sera de même pour l'association **Effectue** qui exige une copie de **No membre** dans la table **Inscription**.



La contrainte inter-association de simultanéité n'a pas d'impact sur le modèle relationnel.

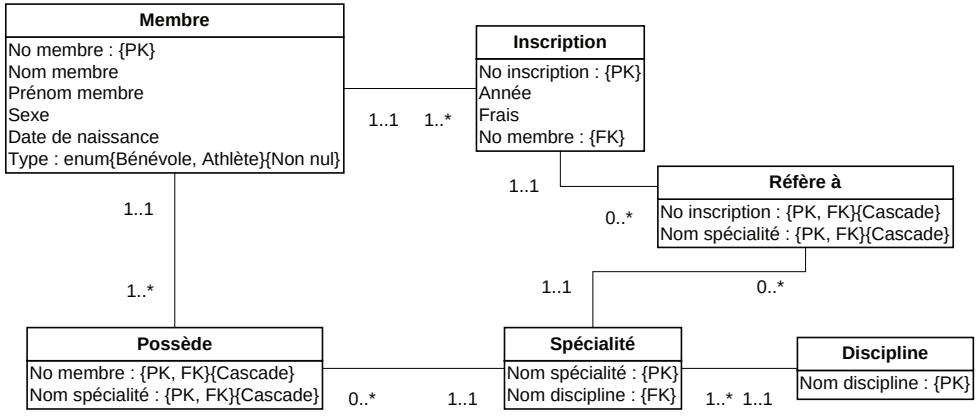
CAS 2-9 CLUB OLYMPIQUE, VERSION *PARTITION*

Traduire ce modèle conceptuel en son pendant relationnel tout en appliquant les règles de dérivation appropriées.



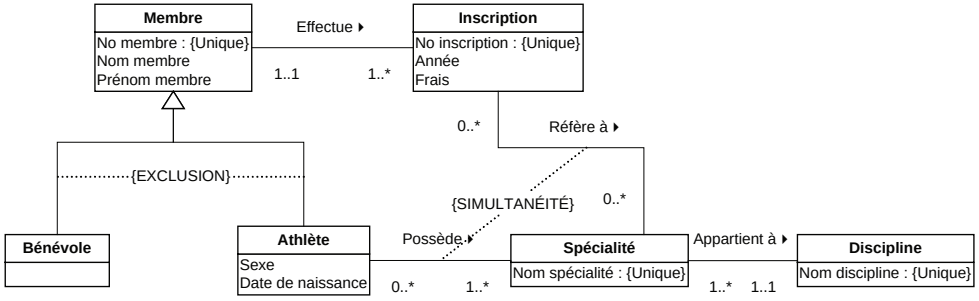
Cette fois le modèle montre des associations d'héritage avec une contrainte de partition. Ce type de contrainte indique que l'entité **Membre** ne peut avoir d'occurrences qui lui soient propres. Les occurrences de **Membre** seront donc soit de type **Bénévole**, soit du type **Athlète** mais JAMAIS LES DEUX À LA FOIS.

Une seule table sera dérivée des associations même si trois entités sont en cause. Elle portera le nom du supertype, comportera comme clé primaire l'identifiant du supertype **Membre**, possédera les attributs combinés de tous les sous-types et un attribut discriminant et obligatoire indiquant le seul type qu'une ligne de cette table pourra avoir.



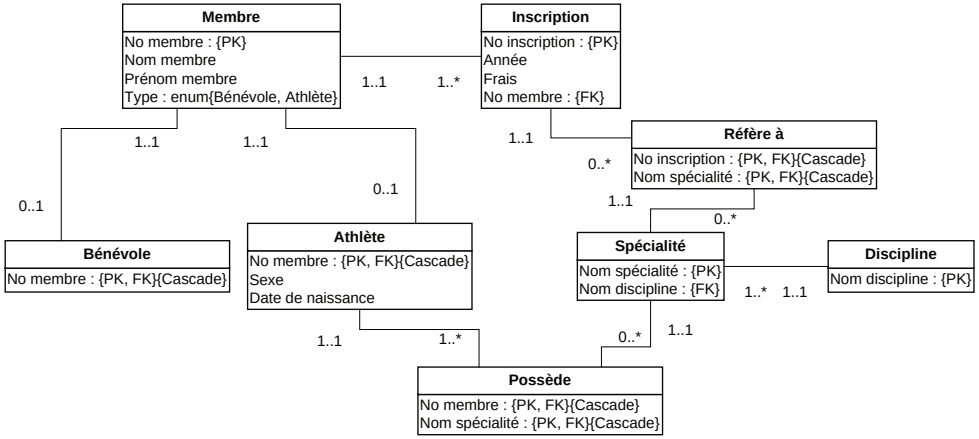
CAS 2-10 CLUB OLYMPIQUE, VERSION *EXCLUSION*

Traduire ce modèle conceptuel en son pendant relationnel tout en appliquant les règles de dérivation appropriées.



Le modèle montre maintenant des associations d'héritage avec une contrainte de partition. Ce type de contrainte indique que l'entité **Membre** PEUT avoir des occurrences qui lui soient propres.

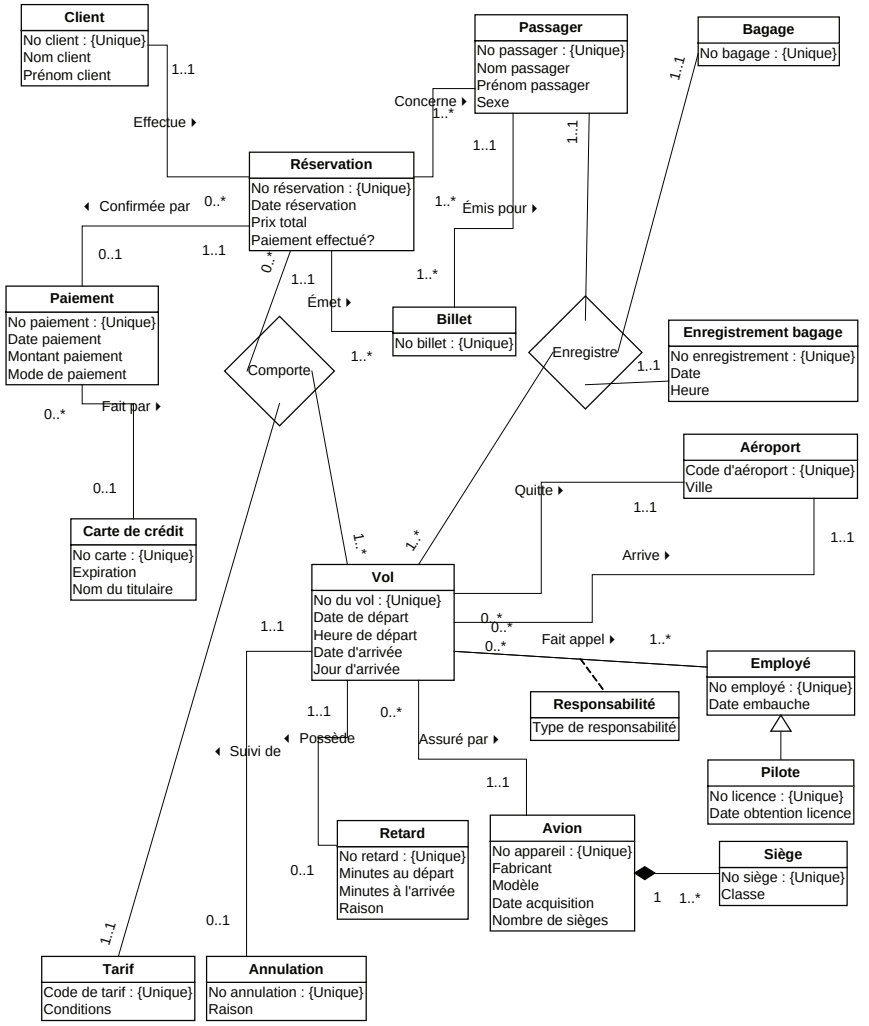
Les occurrences de **Membre** pourraient aussi être de type **Bénévole**, ou du type **Athlète** mais JAMAIS LES DEUX À LA FOIS.



Une table est dérivée de chaque entité. La clé primaire des tables dérivées des sous-types possède comme clé primaire, une copie de la clé primaire du supertype, soit **No membre**. Cette copie est aussi une clé étrangère avec intégrité référentielle en ajout, suppression et mise à jour. Un attribut de la table **Membre**, **Type**, stipule le sous-type auquel une ligne de la table peut appartenir. Les associations liant les tables dérivées de l'association d'héritage portent systématiquement les multiplicités 1..1 et 0..1. 1..1 du côté de la table mère **Membre**.

CAS 2-11 TRANSPORTEUR AÉRIEN

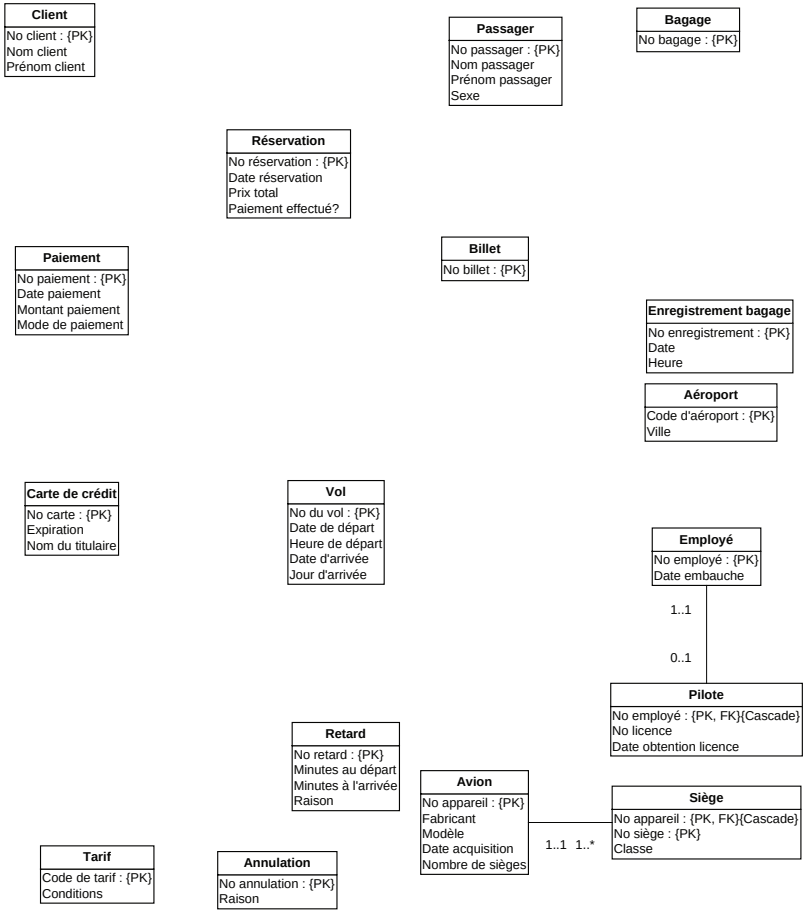
Ce cas, et le cas 2-12 qui suit, démontrent l'importance de décomposer les associations de degré supérieur avant d'en dériver le modèle relationnel. Dans le cadre du présent cas, les associations de degré supérieur ne seront pas décomposées avant d'appliquer les règles de passage au modèle relationnel.



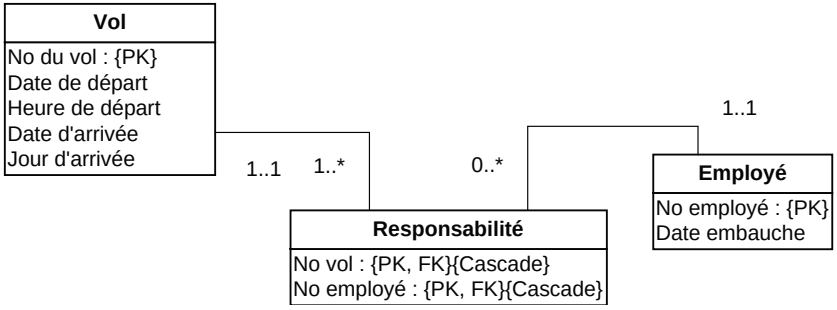
La première association prise en charge est l'association d'héritage. Aucune contrainte ne la concerne. Une table est dérivée de chaque entité. Le supertype **Employé** donne lieu à une table mère et à une association avec la table fille **Pilote** comportant les multiplicités 1..1 et 0..1.

L'association de composition est ensuite considérée. Une table mère **Avion** est dérivée ainsi que sa table fille **Siège**. Les multiplicités sont reprises intégralement du modèle conceptuel.

Les entités non impliquées dans l'héritage et la composition sont toutes des entités fortes sauf **Responsabilité**, une entité d'association. Une première version du modèle relationnel dérivé est montrée ci-après.



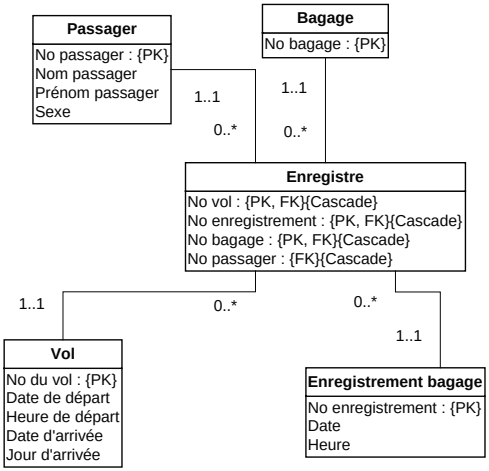
La seule entité d'association, **Responsabilité**, est à son tour analysée. Elle donne lieu à une table fille **Responsabilité** associée à deux tables mères : **Vol** et **Employé**. La clé primaire combine **No vol** et **No employé**. Les multiplicités du côté de la table fille sont ainsi déterminées : les multiplicités à la terminaison d'arrivée pour une lecture à partir de **Vol** sont 1..*, il faut retrouver 1..* du côté de la table fille pour l'association avec la table **Vol** ; les multiplicités à la terminaison d'arrivée pour une lecture à partir de **Employé** sont 0..*, il faut retrouver 0..* du côté de la table fille pour l'association avec la table **Employé**.



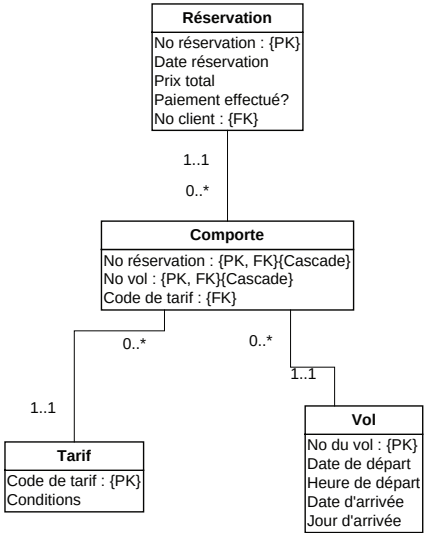
Parmi les associations binaires restantes une seule est du type PLUSIEURS À PLUSIEURS, soit **Concerne** entre **Réservation** et **Passager**. La table fille **Concerne** en est dérivée. Les autres sont du type UN À UN ou UN À PLUSIEURS. On retrouve dans ce modèle plusieurs associations UN À UN où la participation est optionnelle d'un seul côté (multiplicité minimale 0). Rappelons que la table dérivée de l'entité où la multiplicité minimale est 0 agira comme table fille pour cette association UN À UN. C'est donc dans cette table qu'est déposée la clé primaire de la table mère à titre de clé étrangère. C'est le cas de **Paiement** qui sera une table fille pour **Réservation** pour assurer l'association **Confirmée par**.

Les associations de degré supérieur sont finalement considérées. Prenons **Enregistre** en premier lieu. Une table en est dérivée dont la clé primaire est la combinaison des quatre clés primaires des tables associées dont chaque élément est une clé étrangère. Or il existe au moins un arc de l'association avec multiplicité maximale de 1. Choisissons arbitrairement l'arc menant vers **Passager**. On peut donc simplifier la clé primaire composée de la table **Enregistre** en retirant **No passager** de la clé tout en la considérant comme clé étrangère. Même s'il existe d'autres arcs avec multiplicité maximale 1, une seule simplification est permise.

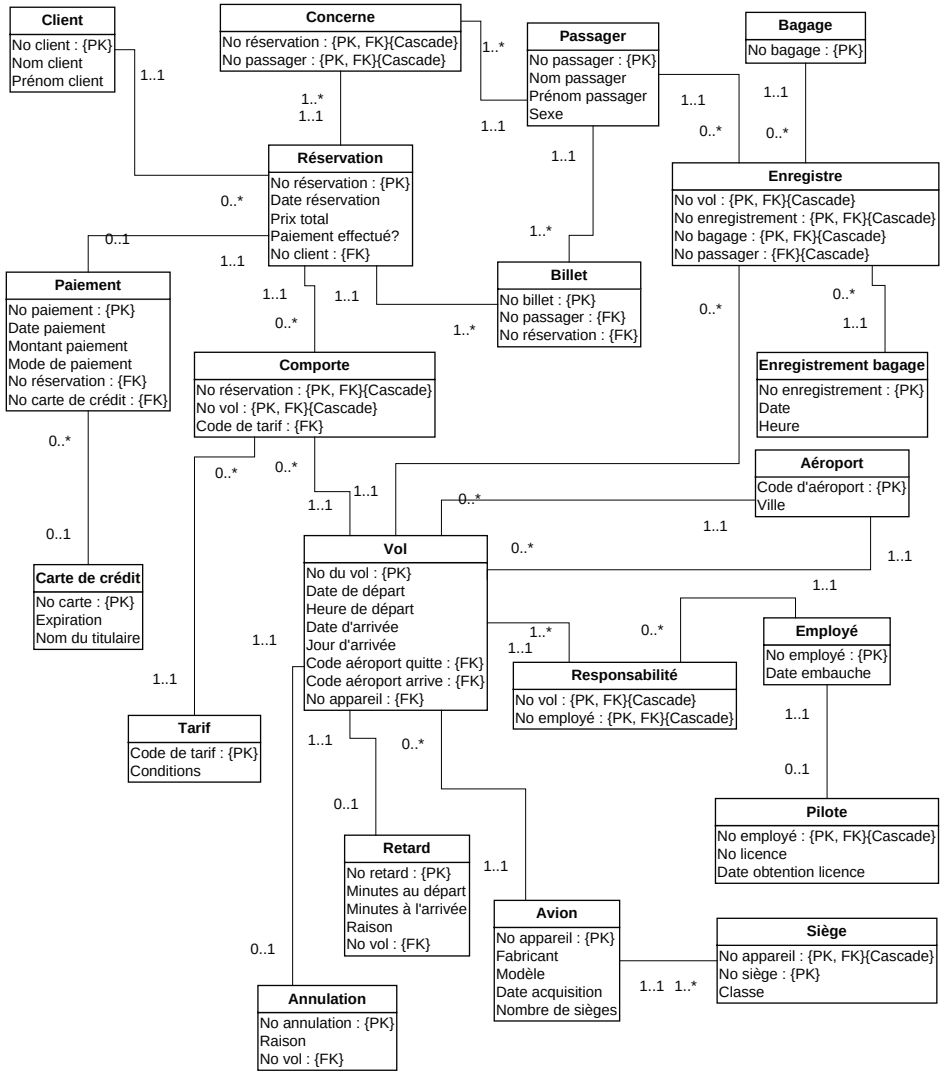
Par ailleurs, toutes les associations de la table **Enregistre** auront les multiplicités 1..1 du côté des tables mères et 0..* d'autre part. C'est la règle pour les associations de degré supérieur. Le modèle dérivé de cette association de degré supérieur est illustré ci-après. On remarquera que l'attribut **No passager** dans la table **Enregistre** ne fait pas partie de la clé primaire à la suite de sa simplification.



Nous appliquons la même démarche de dérivation pour l'association de degré supérieure **Comporte** qui possède elle aussi un arc avec multiplicité maximale 1, l'arc menant à **Tarif**. Ce qui implique que la clé primaire de **Tarif**, **Code de tarif**, peut être retirée de la clé primaire de la table **Compose** qui ne sera formée que des clés primaires des tables **Vol** et **Réservation**. Toutes les associations de la table **Comporte** auront les multiplicités 1..1 du côté des tables mères et 0..* d'autre part. Le modèle dérivé est illustré ci-après.

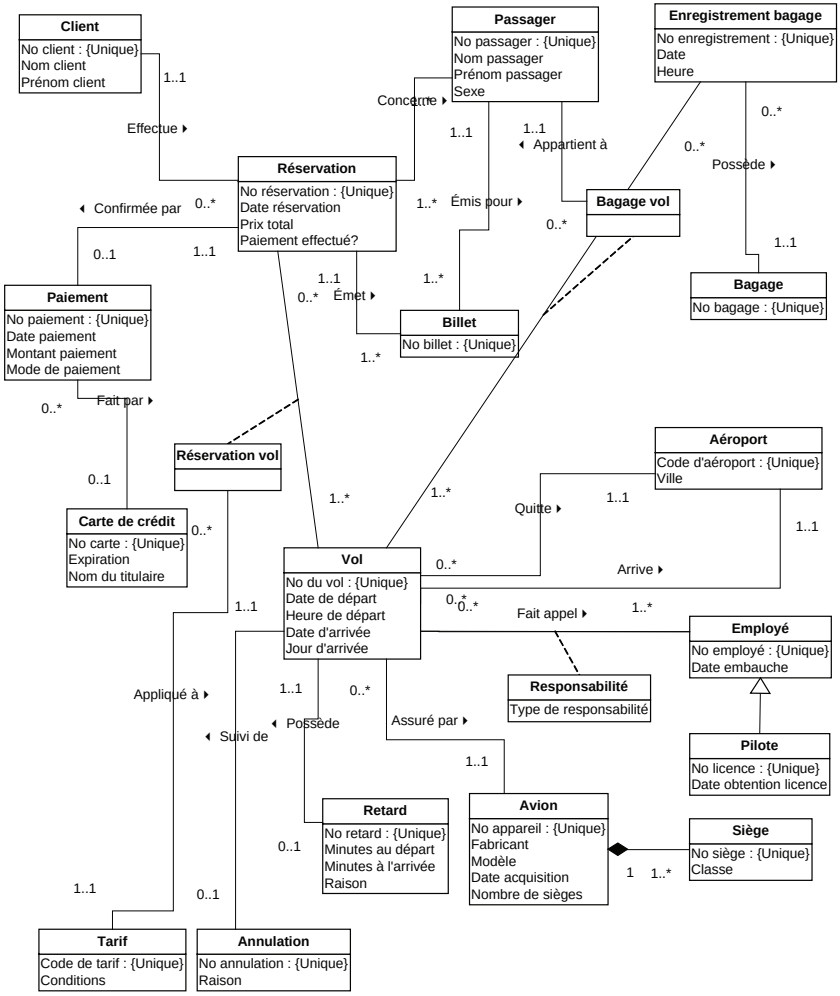


Notons en terminant une particularité du modèle conceptuel 2-11. Les entités **Vol** et **Aéroport** sont associées par deux associations binaires et **Vol** s'avère être table fille dans les deux cas. La clé primaire de **Aéroport** doit en conséquence y être déposée sous deux noms différents à titre de clés étrangères. Le modèle final dérivé du modèle conceptuel est donné ci-après.



CAS 2-12 TRANSPORTEUR AÉRIEN, VERSION 2

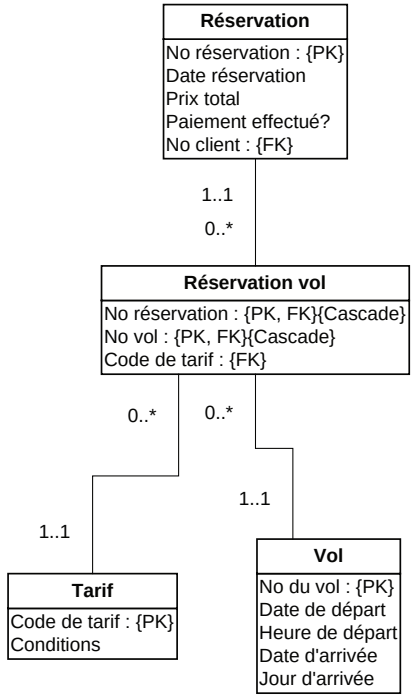
Traduire ce modèle conceptuel en son pendant relationnel tout en appliquant les règles de dérivation appropriées. Puisqu'il s'agit d'une version du modèle conceptuel du cas 2-11 où les associations de degré supérieur ont été décomposées, le modèle relationnel obtenu sera beaucoup plus conforme aux contraintes de multiplicité conceptuelles que celui dérivé de 2-11.



Considérons à tour de rôle les deux différences avec le modèle 2-11.

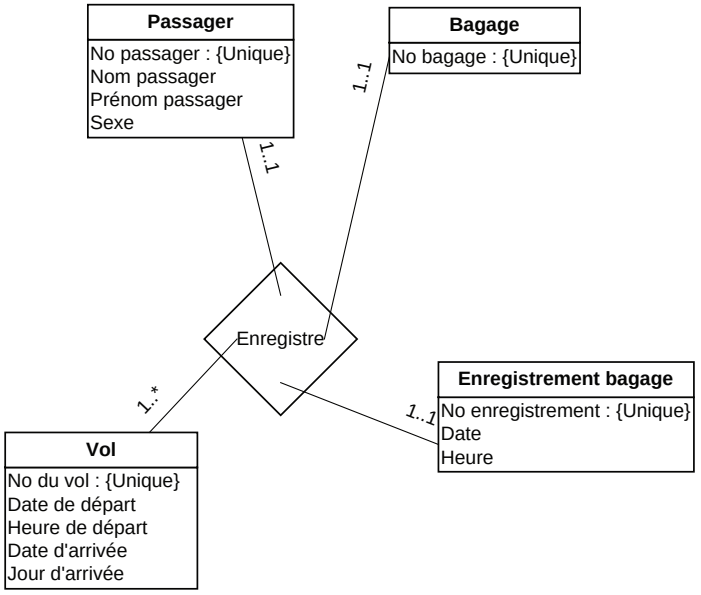
1. L'association de degré supérieur **Comporte** a été décomposée. Nous référons le lecteur au chapitre 1 et en particulier à la figure 1-32 pour une explication détaillée du processus de décomposition. La décomposition conduit à une entité d'association **Réservation Vol** sur une association binaire PLUSIEURS À PLUSIEURS entre **Réservation** et **Vol**. Cette entité d'association est associée à **Tarif** en association binaire UN À PLUSIEURS.

   Une table fille **Réservation Vol** est dérivée pour l'entité d'association, elle sera associée aux tables mères **Réservation** et **Vol**. Pour réaliser la deuxième association binaire, cette fois UN À PLUSIEURS avec **Tarif** et sans entité d'association, la table **Réservation Vol** sera aussi une table fille pour une nouvelle association cette fois avec la table mère **Tarif**. Le modèle relationnel dérivé est le suivant.



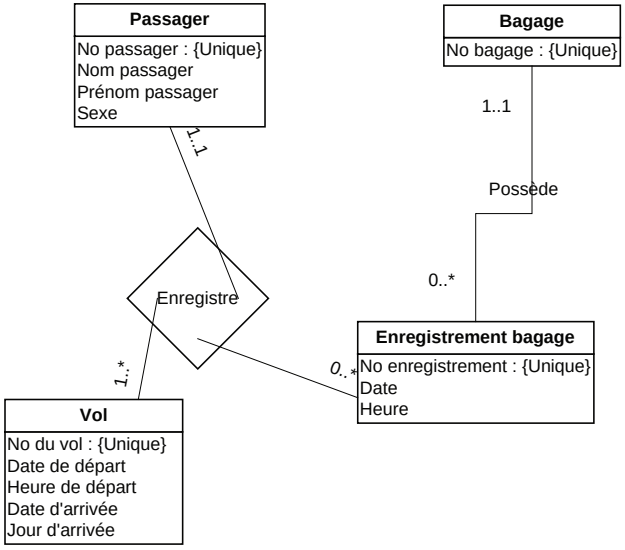
Le lecteur notera que ce modèle est exactement le même que celui dérivé de l'entité de degré supérieur non décomposé, sinon que le nom de la table fille **Réservation vol** est différent de **Comporte**, soit un nom beaucoup plus pertinent que le précédent.

2. L'association de degré supérieur **Enregistre** a été décomposée. Puisque nous n'avons pas expliqué au chapitre 1 le processus de décomposition de cette association, débutons par des explications à ce sujet. Avant décomposition, **Enregistre** est de degré quatre comme illustré ci-après.

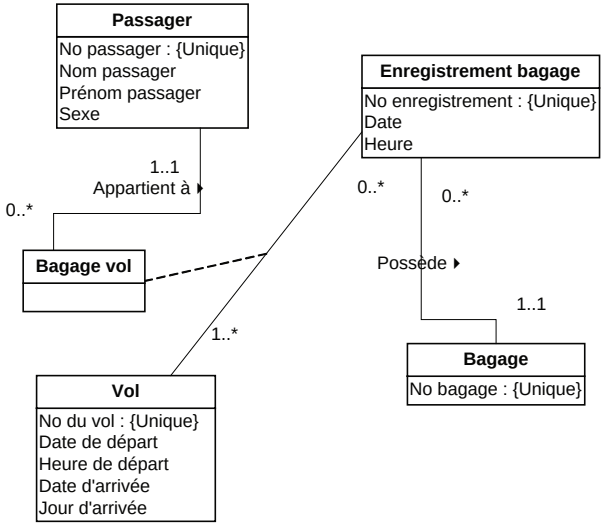


Il y a nettement une dépendance fonctionnelle entre **Enregistrement bagage** et **Bagage**, puisqu'un **No enregistrement** ne correspond qu'à un et un seul bagage. L'entité **Enregistrement Bagage** reste dans l'association, **Bagage** peut en être détachée mais devra être associée directement à **Enregistrement Bagage** par une association UN À PLUSIEURS car le même bagage peut être enregistré plusieurs fois avec des numéros différents.

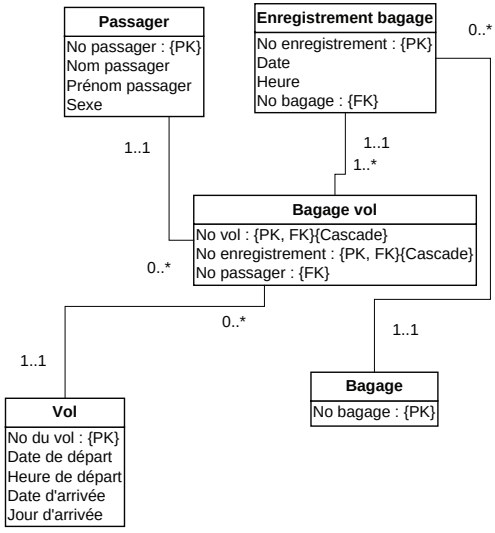
Cette première décomposition mène à une association de degré trois. Les multiplicités de cette association sont toutes réévaluées. Un enregistrement bagage sur un vol est lié à un et un seul passager. Un enregistrement bagage d'un passager doit être sur au moins un vol mais aussi sur plusieurs vols. Un passager sur un vol peut ne pas avoir de bagage ou en avoir plusieurs. La décomposition mène au modèle qui suit.



Une multiplicité maximale 1 demeure du côté de **Passager**. Il n'y a pas de dépendance fonctionnelle de cette entité avec les deux autres. Une entité d'association appelée **Bagage vol** sans attributs va être créée et elle sera associée par ailleurs à **Passager**. Les multiplicités sont à nouveau réévaluées. Le résultat est le suivant.



On peut maintenant procéder à la dérivation des tables pour ce fragment du modèle conceptuel. L'entité d'association donne lieu à une table fille associée à **Vol** et **Enregistrement bagage**, appelée **Bagage vol**. Elle est par ailleurs associée à la table **Passager** pour laquelle elle est une table fille. Enfin, **Enregistrement bagage** doit être une table fille pour la table **Bagage**. La dérivation produit en conséquence le modèle relationnel suivant.

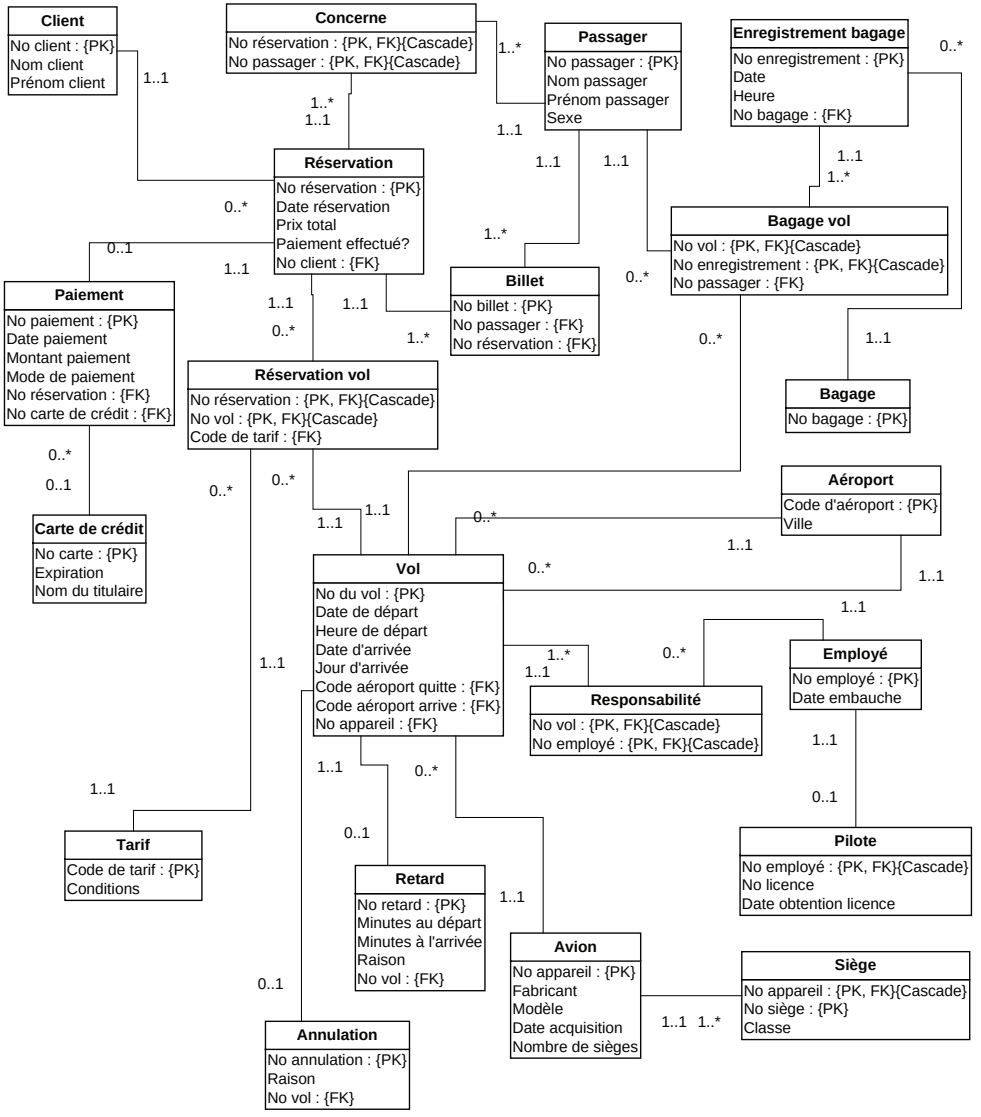


Cette fois le modèle dérivé de l'entité d'association de degré quatre **Enregistre** tiré du cas 2-11 est complètement différent de celui obtenu après décomposition complète de la même association dans le cas actuel. Le modèle relationnel final donné ci-après est beaucoup plus conforme au modèle conceptuel car toutes les multiplicités ont pu être exprimées.

Rappelons que dans le cas 2-11, lors de la dérivation de la table fille **Enregistre**, les multiplicités du côté de celle-ci ont systématiquement été établies à 0..* et celles du côté des tables mères à 1..1, sans égard aux multiplicités présentes sur les arcs de l'association. On n'a donc pas été en mesure de dériver fidèlement toutes les multiplicités du modèle conceptuel.

Comme nous l'indiquions au cas 2-7, la raison en est que la sémantique des multiplicités d'une association de degré supérieure en UML n'a pas son équivalent dans le modèle relationnel. Il s'agit d'une preuve supplémentaire de l'importance de décomposer toutes les associations de degré supérieur, si la chose est possible, avant de dériver le modèle relationnel.

La prochaine illustration montre la version finale du modèle relationnel dérivé.



MODÈLE RELATIONNEL DE DONNÉES NORMALISÉ

Traditionnellement, dans la démarche menant à la réalisation d'une base de données relationnelle, beaucoup d'importance est accordée à la production d'un schéma relationnel *normalisé*. Nous allons montrer sous cette rubrique que l'application des règles de modélisation vues au chapitre un conduisent à un modèle relationnel normalisé, donc à un schéma relationnel normalisé.

En vertu de la théorie de normalisation, un schéma relationnel normalisé doit répondre aux exigences minimales suivantes :

- ***Non redondance***; un attribut n'appartient qu'à une seule relation, donc à une seule table, à moins qu'il n'agisse comme clé étrangère pour assurer l'association avec une autre table;
- ***Cohérence***; les attributs qui décrivent le même objet appartiennent à la même table et dépendent chacun fonctionnellement et totalement de la clé primaire de la table.

Pour assurer l'exigence de cohérence, chaque table doit répondre à trois propriétés de base appelées les trois formes normales :

- La première forme normale, notée 1NF;
- La deuxième forme normale, notée 2NF;
- La troisième forme normale, notée 3NF.

Première forme normale

Une relation, donc une table, est en première forme normale si chacun de ses attributs ne comporte qu'une et une seule valeur. Cela revient à dire que sa valeur est atomique et répond à un type de données simple (*First normal form*).

Cette propriété est assurée dans le modèle conceptuel par la *règle de description*. En conséquence, si la règle de description est appliquée à tous les attributs du modèle conceptuel, les tables du modèle relationnel répondent toutes à la première forme normale.

Deuxième forme normale

Une relation, donc une table, est en deuxième forme normale si elle est d'abord en première forme normale. De plus, tout attribut hors de la clé primaire doit dépendre fonctionnellement et totalement de la clé primaire (*Second normal form*).

Dépendance fonctionnelle entre deux attributs

Un attribut **B** dépend fonctionnellement de l'attribut **A**, noté $\mathbf{A} \rightarrow \mathbf{B}$, si à une valeur de **A** correspond une et une seule valeur pour **B**. On dit que **A** détermine **B** (*Functional dependency*).

Si l'identifiant d'une entité est simple, par définition sa valeur détermine une et une seule valeur pour chacun des autres attributs. En conséquence avec un choix d'identifiant *simple* approprié dans le modèle conceptuel, toutes les entités fortes produisent une table dont chaque attribut dépend fonctionnellement de la clé primaire dérivée de l'identifiant. La deuxième forme normale est assurée pour les tables dérivées d'une entité forte avec clé primaire simple incluant les tables dérivées d'une association d'héritage dont le supertype possède un identifiant simple.

Considérons maintenant le cas des entités fortes à clé primaire composée. Rappelons que par convention l'identifiant composé doit être minimal, c'est à dire que l'identifiant doit regrouper le plus petit nombre d'attributs qui permettent de déterminer TOUS les autres attributs. Chaque attribut hors de l'identifiant dépend alors fonctionnellement d'un groupe minimal d'attributs. Considérons maintenant une définition élargie de la dépendance fonctionnelle.

Dépendance fonctionnelle entre un attribut et un groupe d'attributs

Un attribut **B** dépend fonctionnellement de **n** attributs $\mathbf{A_1,..,A_n}$, noté $\mathbf{(A_1,..,A_n) \rightarrow B}$, si à un tuple de valeurs $\mathbf{(V_1,..,V_n)}$ dont chaque élément provient dans l'ordre de $\mathbf{A_1}$ **à** $\mathbf{A_n}$, correspond une et une seule valeur pour **B**. On dit que $\mathbf{(A_1,..,A_n)}$ détermine **B** (*Functional dependency*).

Illustrons la dépendance fonctionnelle entre un attribut et un groupe d'attributs à l'aide d'un exemple. Considérons l'attribut **Date d'obtention diplôme**. Cet attribut dépend à la fois de **No étudiant** et **Nom diplôme**, car le même diplôme ne peut être accordé deux fois au même étudiant. On dira qu'un couple de valeurs (**LEGP27059001**, **'Bac. En administration des affaires'**) détermine **'27 mai 2006'** car l'étudiant portant le numéro **LEGP27059001** a obtenu son diplôme **Bac. En administration des affaires** le **27 mai 2006**.

Nous retrouvons en conséquence une dépendance fonctionnelle entre l'attribut **Date d'obtention diplôme** et un groupe de deux attributs (**No étudiant** et **Nom diplôme**).

Si un attribut **A**, hors de l'identifiant composé d'une entité forte dépend aussi fonctionnellement d'un sous-ensemble des attributs de l'identifiant, le groupe minimal d'attributs qui détermine **A** n'est pas le même que pour les autres. C'est un indice que cet attribut n'est pas à sa place et qu'il y a transgression de la *règle de construction* qui stipule qu'une entité ne peut contenir une autre entité ou les attributs d'une autre entité. C'est aussi une condition qui fait que la table dérivée de l'entité contenant **A** et les autres attributs n'est pas en deuxième forme normale.

Reprenons l'exemple précédent. Considérons les attributs **Date d'obtention diplôme** et **Nom étudiant**. Nous retrouvons, tout comme pour **Date d'obtention diplôme**, une dépendance fonctionnelle entre **Nom étudiant** et un groupe de deux attributs (**No étudiant** et **Nom diplôme**).

On dira que le couple de valeurs (**'Pierre Léger'**, **'Bac. En administration des affaires'**) détermine **'27 mai 2006'** car l'étudiant **Pierre Léger** a obtenu son diplôme **Baccalauréat en administration des affaires** le **27 mai 2006**.

Mais **Nom étudiant** dépend aussi fonctionnellement de **No étudiant**, car **'Pierre Léger'** n'a qu'un seul numéro d'étudiant comme tous les étudiants d'ailleurs. Cela signifie que **Date d'obtention diplôme** et **Nom étudiant** n'ont pas le même identifiant minimal. En vertu de la règle de construction, ils ne décrivent pas le même objet. Ils doivent être présents dans deux entités distinctes.

La deuxième forme normale est assurée pour les tables dérivées d'une entité forte avec clé primaire composée si l'identifiant est minimal et que cette entité respecte la règle de construction.

Il nous reste à considérer maintenant les entités faibles. En matière de modélisation conceptuelle, une entité d'association ne peut exister que si elle n'a pas d'identifiant propre et si ses attributs dépendent individuellement de la combinaison des identifiants des entités associées. Cette définition stricte d'une entité d'association vise précisément à assurer que la table dérivée d'une entité d'association est en deuxième forme normale. Il en va de même pour une entité composant dont l'identifiant propre est combiné à l'identifiant du composite pour donner la clé primaire de la table dérivée du composant où une dépendance fonctionnelle totale de chaque attribut est assurée avec la clé primaire composée.

Considérons un modèle conceptuel comportant une entité **Étudiant** avec comme identifiant **No étudiant** et une entité **Diplôme** dont l'identifiant est **Nom diplôme**. Nous avons vu plus tôt que **No étudiant** et **Nom diplôme** déterminent conjointement **Date d'obtention diplôme**. Cet attribut ne pourrait en conséquence qu'appartenir à une entité d'association sur une association *plusieurs à plusieurs* entre les entités **Étudiant** et **Diplôme** car son identifiant est en fait la combinaison des identifiants de **Étudiant** et **Diplôme**. En effet, chaque attribut de l'identifiant pris isolément ne peut déterminer qu'une seule date d'obtention de diplômes : un étudiant peut avoir plusieurs diplômes, donc plusieurs dates d'obtention de diplômes et le même type de diplôme peut être accordé à plusieurs étudiants.

Bref, les conventions relatives au choix d'un bon identifiant pour les entités fortes, les conventions justifiant l'existence d'une entité d'association tout comme une entité composant ainsi que la règle de construction, mènent d'emblée à un modèle conceptuel où la deuxième forme normale est assurée.

Troisième forme normale

Une relation, donc une table, est en troisième forme normale si elle est d'abord en deuxième forme normale. De plus, chaque attribut hors de la clé primaire ne peut dépendre fonctionnellement d'un attribut ou d'un groupe d'attributs hors de la clé primaire (*Third normal form*).

La *règle de construction* permet d'éviter une telle situation. Si deux attributs hors de l'identifiant dépendent l'un de l'autre, c'est qu'ils appartiennent manifestement à une deuxième entité, transgressant ainsi la règle de construction. Le modélisateur doit considérer deux entités plutôt qu'une seule. Les tables dérivées de ces deux entités seront en troisième forme normale si elles sont aussi en deuxième forme normale.

En résumé, un modèle relationnel dérivé d'un modèle conceptuel est normalisé dans le modèle conceptuel si et seulement si :

- la règle de *non redondance* est respectée ;
- le choix d'un identifiant composé est minimal et le même pour tous les attributs hors de l'identifiant d'une même entité ;
- la règle de *description* est respectée pour chaque entité ;
- la règle de *construction* est respectée pour chaque entité ;
- les entités d'association ont un identifiant implicite dont dépend totalement ses attributs ;
- l'identifiant implicite est la combinaison des identifiants de toutes les entités associées ;
- les entités composants ont un identifiant qui permet de distinguer chaque occurrence rattachée au même composite.

OPTIMISATION DU MODÈLE RELATIONNEL

Le modèle relationnel peut être optimisé de manière à produire un modèle physique techniquement performant. Deux mesures affectant le modèle relationnel peuvent être considérées de façon à améliorer la performance du modèle physique qui en sera dérivé.

1. Le nombre de tables peut être réduit en fusionnant certaines tables pour en arriver à éviter des opérations de jointure inutiles ;

2. Une clé primaire composée peut être remplacée par une clé primaire simple pour réduire la complexité des index et rendre plus performantes les opérations de jointure.

Il peut être difficile pour le lecteur de voir à ce stade-ci la portée et la pertinence des mesures d'optimisation proposées sous cette rubrique. Nous aurons l'occasion au chapitre 3 de les justifier davantage.

Deux types d'associations binaires présentes dans un modèle relationnel peuvent entraîner la fusion des deux tables concernées. Les associations *un à un* dont les multiplicités minimales sont 1 et 1. Les associations *un à un* dont les multiplicités minimales sont 0 et 1.

Association binaire avec multiplicités 1..1 – 1..1

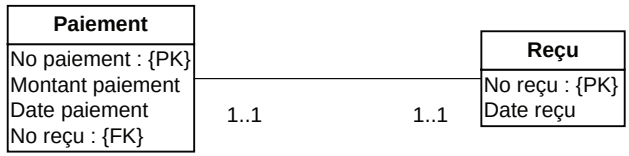
Si dans une association *un à un* la participation est obligatoire des deux côtés de l'association (multiplicités 1..1 de part et d'autre), les attributs des deux tables peuvent être fusionnés pour former une seule table. La table qui disparaît est celle dont la clé primaire est une clé étrangère dans la deuxième table. Tous ses attributs sont ajoutés à la seule table restante, incluant les clés étrangères présentes dans la table éliminée, sauf sa clé primaire déjà présente à l'autre table à titre de clé étrangère. Dans le cas où des attributs portent le même nom dans les tables fusionnées, leur nom doit être changé pour éviter toute confusion.

La clé primaire de la table qui a disparu suite à la fusion ne doit plus porter la mention clé étrangère {FK} dans la table résultant de la fusion. La mention est impérativement remplacée par les contraintes d'intégrité *{Non nul, Unique}*. Cet attribut, maintenant doté des contraintes *Non nul* et *Unique* peut être considéré comme une *clé secondaire*, soit un attribut ou un groupe d'attributs qui pourrait aussi jouer le rôle d'une clé primaire.

Clé secondaire ▸ Un attribut ou groupe d'attributs qui a les propriétés d'une clé primaire donc répondant à la contrainte d'intégrité d'entité {Non nul, Unique} (*Secondary key*).

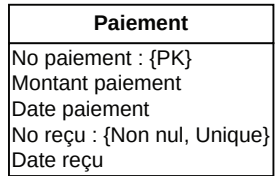
Considérons la figure 2-34 où une association binaire *un à un* montre des multiplicités 1..1 de part et d'autre des entités concernées.

FIGURE 2-34 **Association binaire *un à un* avec participation obligatoire de part et d'autre**



No reçu est une clé étrangère dans la table **Paiement**. La table **Reçu** sera donc intégrée à **Paiement** et la mention {FK} sur l'attribut **No reçu** remplacée par les contraintes d'intégrité *{Non nul, Unique}* qui correspondent à la contrainte de multiplicité 1..1.

FIGURE 2-35 **Table dérivée du modèle 2-34**



Dans le cas où la table qui disparaît possède des associations avec d'autres tables, ces associations sont maintenues en les rattachant à la table résultant de la fusion. Si la clé primaire de la table éliminée **No reçu** agit comme clé secondaire pour la table résultant de la fusion, sa présence en tant que clé étrangère dans les tables associées demeure pertinente.

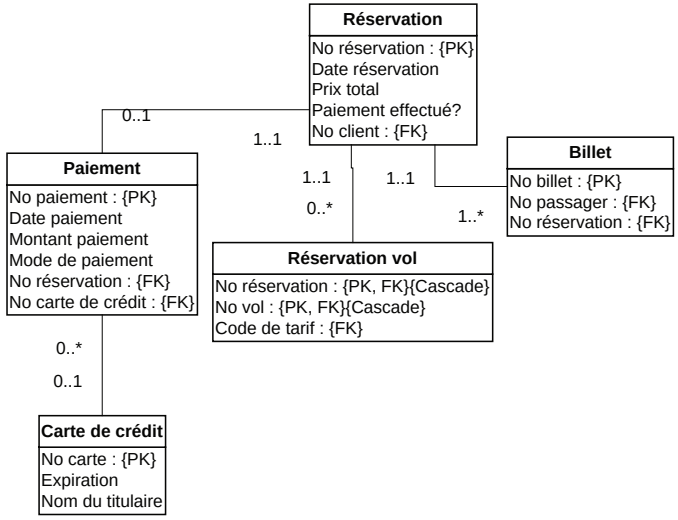
Association binaire avec multiplicités 0..1 – 1..1

Lorsque dans une association *un à un* la participation est obligatoire d'un seul côté de l'association (multiplicités 0..1 d'une part et 1..1 de l'autre), les attributs des deux tables peuvent être fusionnés pour former une même table. La table qui disparaît est celle du côté 0..1 et ses attributs sont ajoutés à l'autre table, incluant les clés étrangères, sauf la clé étrangère qui est la clé primaire de l'autre table pour éviter de dupliquer la clé primaire. Dans le cas où des attributs portent le même nom dans les tables fusionnées, leur nom doit être changé.

Toute table associée à la table éliminée le sera désormais avec la table résultant de la fusion. Dans le cas où la table éliminée est une table mère, la clé primaire déposée à titre de clé étrangère dans chaque table fille doit être remplacée par la clé primaire de la table résultant de la fusion.

La clé primaire de la table qui a disparu suite à la fusion ne doit plus porter la mention clé primaire {PK} dans la table résultant de la fusion. La mention est impérativement remplacée par la contrainte d'intégrité {Unique} qui correspond à la contrainte de multiplicité 0..1. Considérons la figure 2-36 où apparaît une association binaire *un à un* entre les tables **Paiement** et **Réservation** avec des multiplicités 0..1 du côté de **Paiement**.

FIGURE 2-36 **Association binaire *un à un* avec participation obligatoire d'un seul côté**

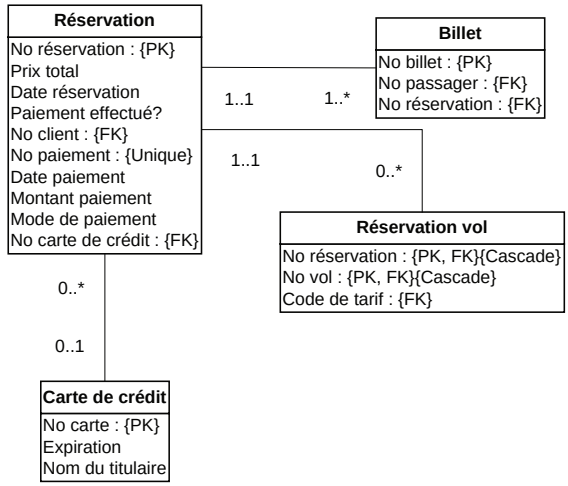


Les attributs de la table **Paiement** seront intégrés à la table **Réservation** sans dupliquer **No réservation**. **No réservation** sera donc la clé primaire de la table résultante. **No paiement** perd sa mention {PK} mais on lui impose la contrainte {Unique} car un numéro de paiement est associé à au plus une réservation.

Toute table qui était associée à **Paiement** le sera désormais à **Réservation**. La table **Carte de crédit** qui était associée à **Paiement** sera donc associée à **Réservation**. Puisqu'elle n'était pas une table fille de **Paiement**, aucune clé étrangère ne doit être changée. Si la table **Carte de crédit** était une table fille

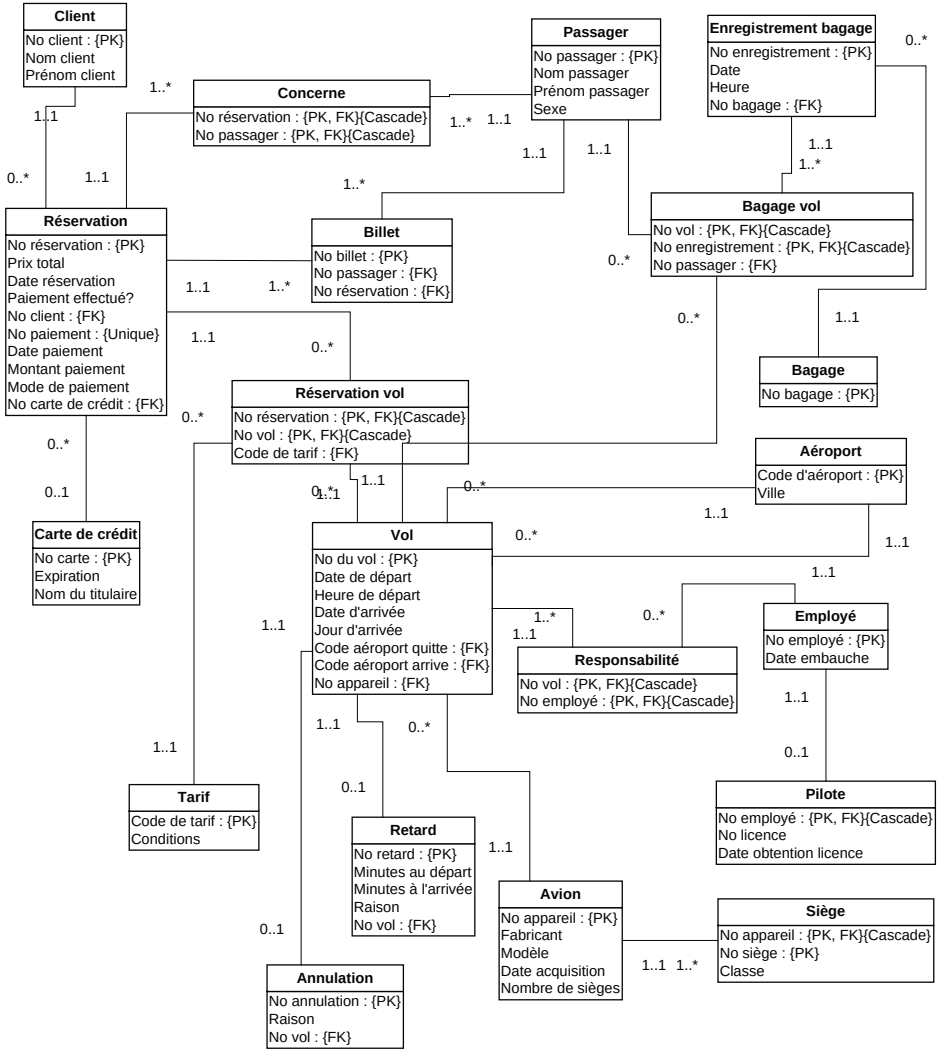
de **Paiement** avant la fusion, elle serait dotée d'un attribut clé étrangère **No paiement** {FK} qui devrait être remplacé par **No réservation** {FK} dans le contexte de la fusion.

FIGURE 2-37 **Résultat de la fusion des tables *Paiement* et *Réservation***



CAS 2-13 TRANSPORTEUR AÉRIEN, MRD PARTIELLEMENT OPTIMISÉ

Pour le modèle relationnel suivant, partiellement optimisé à la suite de la fusion des tables **Paiement** et **Réservation**, il s'agit de poursuivre la démarche d'optimisation en éliminant les associations binaires 0..1-1..1 présentes.



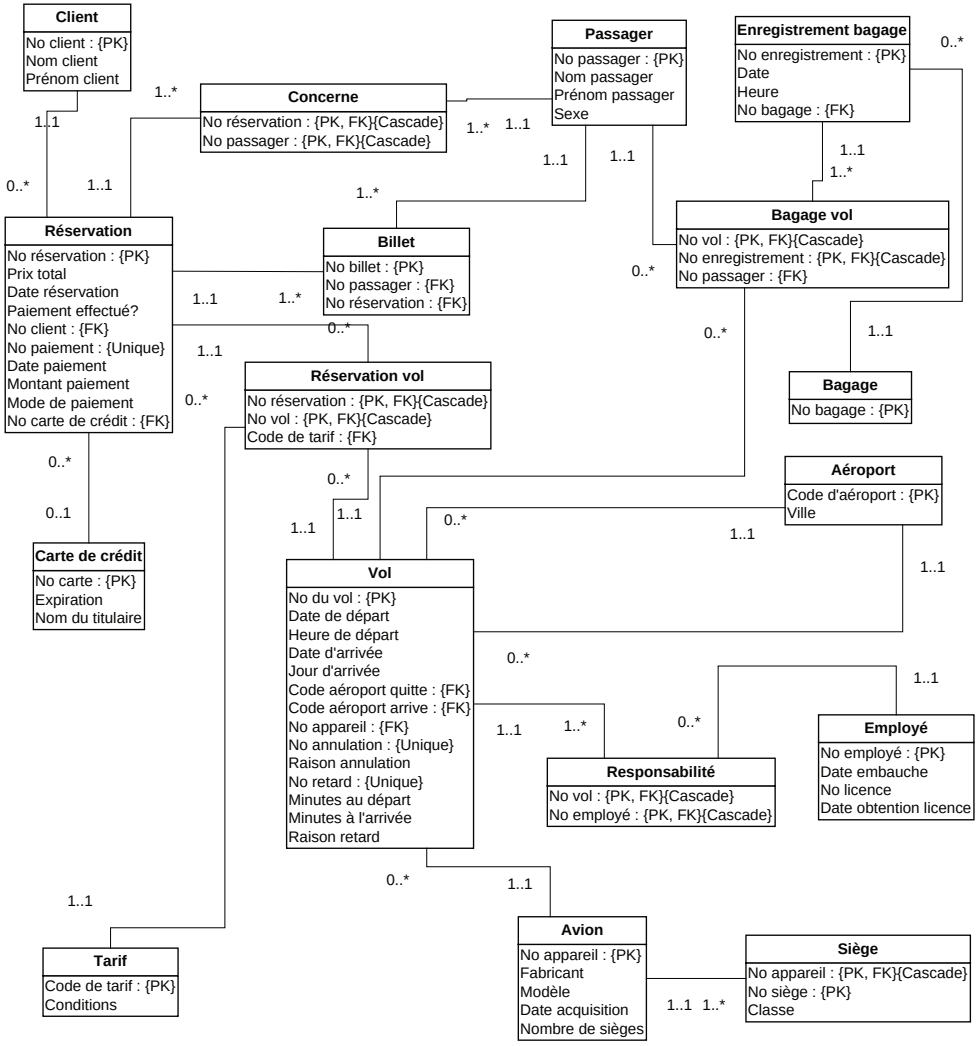
Débutons le processus de fusion avec l'association entre **Annulation** et **Vol**. La table **Annulation**, placée du côté 0..1 de l'association, devrait être retirée. Ses attributs, sauf **No vol** qui est la clé primaire de **Vol**, doivent être ajoutés à la table **Vol** et la mention {PK} de sa clé primaire est remplacée par la contrainte {Unique}. La nouvelle table **Vol** est donc la suivante.

Vol
No du vol : {PK}
Date de départ
Heure de départ
Date d'arrivée
Jour d'arrivée
Code aéroport quitte : {FK}
Code aéroport arrive : {FK}
No appareil : {FK}
No annulation : {Unique}
Raison

Considérons maintenant l'association entre **Retard** et **Vol**. La table **Retard**, placée du côté 0..1 de l'association, devrait être retirée. Ses attributs, sauf **No vol** qui est la clé primaire de **Vol**, doivent être ajoutés à la table **Vol** et la mention {PK} de sa clé primaire est remplacée par la contrainte {Unique}. Cette fois-ci on note la présence d'un attribut **Raison**, présent à la fois dans la table **Retard** et la table **Vol**. Il est nécessaire de changer leur nom pour éviter toute confusion. Ils seront renommés **Raison annulation** et **Raison retard**. La nouvelle table **Vol** est maintenant la suivante.

Vol
No du vol : {PK}
Date de départ
Heure de départ
Date d'arrivée
Jour d'arrivée
Code aéroport quitte : {FK}
Code aéroport arrive : {FK}
No appareil : {FK}
No annulation : {Unique}
Raison annulation
No retard : {Unique}
Minutes au départ
Minutes à l'arrivée
Raison retard

Quant à la dernière association binaire 0..1-1..1 qui demeure, elle concerne les tables **Employé** et **Pilote**. La table **Pilote**, placée du côté 0..1 de l'association, devrait être retirée. Ses attributs, sauf **No employé** qui est la clé primaire de **Employé**, doivent être ajoutés à la table **Employé**. Le modèle relationnel résultant de ces étapes d'optimisation est donné ci-après.



Simplification des clés primaires

Les bases de données disposent de ce que l'on appelle des *index* pour permettre un accès rapide à chaque enregistrement à partir de sa clé. Dans le cas des bases de données relationnelles, chaque table est dotée d'au moins un index de manière à localiser une ligne de la table à partir de la valeur de la clé primaire. Les index sont des composants physiques de la base de données dont on n'a pas à se préoccuper dans le modèle relationnel. Néanmoins il y a des choix faits par le modélisateur aux niveaux conceptuel et relationnel qui ont un impact majeur sur la performance de ces index, compte tenu notamment de leur taille et de leur complexité.

Une clé primaire composée exige un index plus complexe et donc moins efficace qu'une clé primaire simple. Une clé primaire composée de taille importante dont une composante est de type `String` produit un index dont la gestion peut être lourde. On considère aussi qu'une clé primaire simple de type `String` et de taille importante, plus de 20 caractères par exemple, induit une redondance dans la base de données qui pourrait être évitée.

Nous allons considérer ci-après deux formes d'optimisation portant sur la clé primaire des tables du modèle relationnel :

1. Substituer une clé primaire simple à une clé primaire composée ;
2. Substituer une clé primaire simple de type numérique à une clé primaire simple de type `String`.

Avant d'aborder à tour de rôle ces deux thèmes, revenons un instant sur une astuce introduite plus tôt dans ce chapitre concernant la validité de la clé primaire composée d'une table dérivée du modèle conceptuel.

Utilisation d'une clé primaire simple avec génération automatique de valeurs séquentielles

Nous avons déjà suggéré l'emploi d'une clé primaire simple portant la mention {PK auto} signifiant par là que le SGBD générera automatiquement, lors de l'ajout d'une nouvelle ligne dans la table, une valeur pour la clé primaire systématiquement différente des autres lignes de la table. Lorsque le modélisateur doute de l'unicité d'accès par la clé primaire composée de la table dérivée d'une association binaire *plusieurs à plusieurs* sans entité d'association ou d'une association de degré supérieur sans entité d'association ne comportant que des multiplicités maximales *plusieurs*, l'astuce consiste à donner à la table fille dérivée une clé primaire simple artificielle portant la mention {PK auto}.

Cette astuce peut être étendue, en toute généralité, aux tables comportant une clé primaire composée, de manière à réduire la complexité des index. Les tables dérivées d'une association *plusieurs à plusieurs* ou d'une association de degré supérieur sont toutes dotées d'une clé primaire composée qui peut être remplacée par une clé primaire simple à génération automatique de valeurs séquentielles, tout en conservant les attributs qui constituaient la clé primaire composée originelle.

Table dérivée d'une association plusieurs à plusieurs ou d'une association de degré supérieur

Les tables de cette nature disposent toutes d'une clé primaire composée de deux attributs ou plus. La technique d'optimisation consiste à donner une nouvelle clé primaire simple dotée de la mention {PK auto}, tout en conservant les attributs de la clé primaire composée et les mentions {FK} et {Cascade}.

Considérons à la figure 2-38 la table **Bagage vol** présente dans le modèle relationnel tiré du cas 2-13.

FIGURE 2-38 **Table tirée du cas 2-13**

Bagage vol
No vol : {PK, FK}{Cascade} No enregistrement : {PK, FK}{Cascade} No passager : {FK}

Elle est dotée d'une clé primaire double **(No vol, No enregistrement)** qu'on a le loisir de remplacer par une clé primaire à génération automatique de valeurs, **No bagage vol**. Le nom de la clé sera de préférence le mot **Numéro** suivi du nom de la table. La nouvelle table conserve les attributs constituant la clé primaire composée originale avec les mentions {FK} et {Cascade} comme le montre la figure 2-39.

FIGURE 2-39 **Table optimisée grâce à une clé primaire simple {PK auto}**

Bagage vol
No bagage vol : {PK auto} No vol : {FK}{Cascade} No enregistrement : {FK}{Cascade} No passager : {FK}

La même technique d'optimisation peut être appliquée aux tables **Concerne**, **Réservation vol** et **Responsabilité** de manière à leur donner une clé primaire simple. Le résultat de telles substitutions est donné à la figure 2-40.

FIGURE 2-40 **Modèle tiré du cas 2-13 où les clés primaires ont été optimisées**

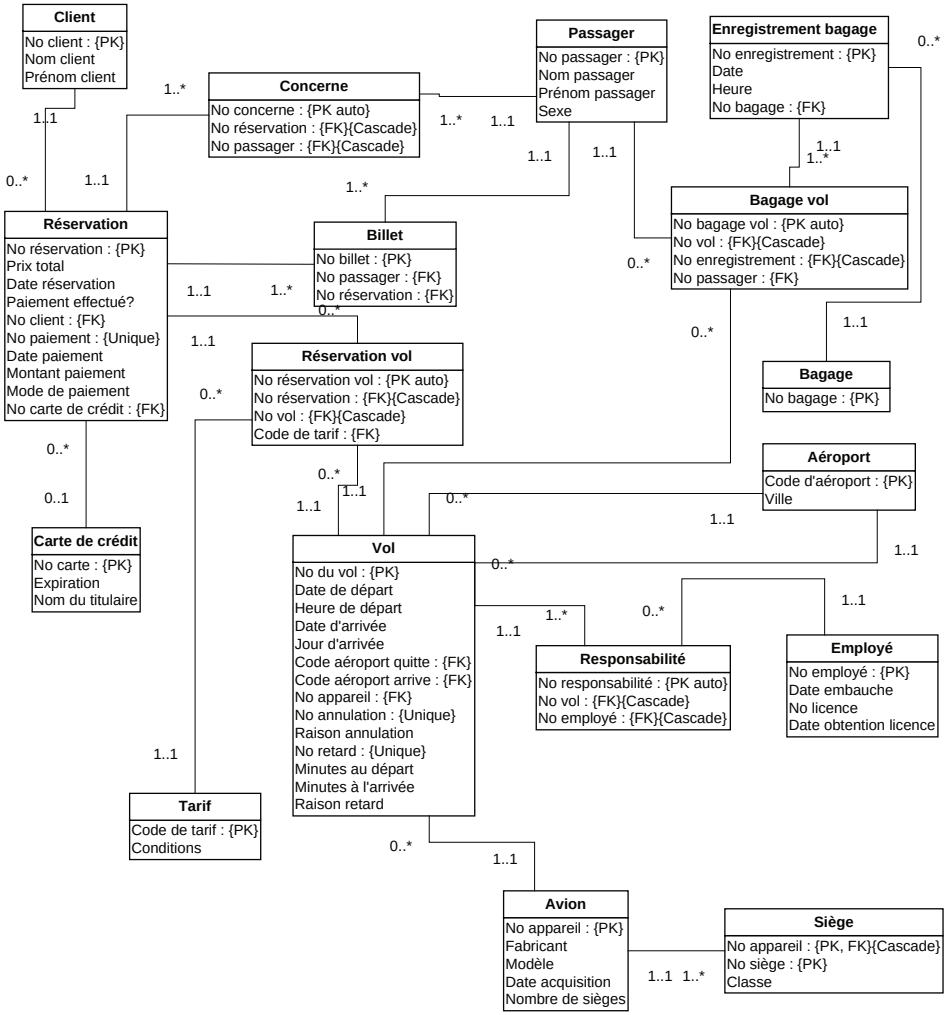
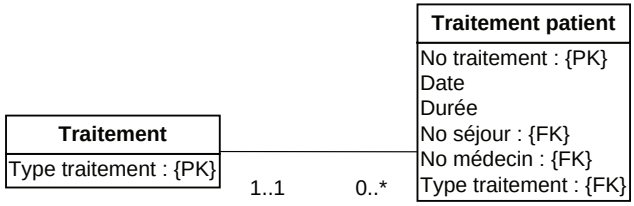


Table dont la clé primaire simple est de type texte

Lorsque la clé primaire d'une table est de type `String` et de taille importante et que de plus elle agit comme table mère pour une ou plusieurs tables, la présence de doubles de cette clé à titre de clé étrangère dans les tables filles constitue une redondante qui peut être optimisée. C'est le cas fréquemment des tables tirées des entités de description.

Les tables de la figure 2-41 sont présentes dans le modèle du cas 2-5. On note la présence de la table mère **Traitement** découlant manifestement d'une entité d'association dont la clé primaire simple est de type `String` et qui est présente comme clé étrangère dans la table **Traitement patient**. On constate que l'attribut **Type traitement** peut posséder une valeur textuelle pouvant dépasser 20 caractères. Par exemple ***'Polypectomie à l'intestin grêle par endoscopie'***.

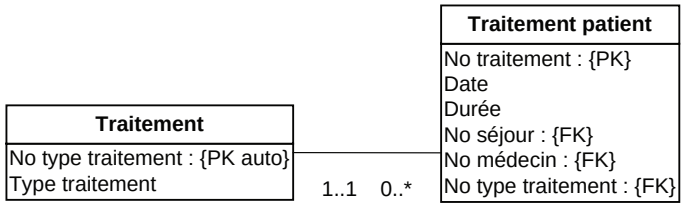
FIGURE 2-41 **Tables tirées du cas 2-5**



La valeur de la clé étrangère constitue une redondance acceptable dans le modèle relationnel. Cette redondance a cependant un impact sur la taille de la base de données et sur la performance des opérations de jointure entre les tables.

Le modélisateur peut juger utile de doter la table **Traitement** d'une clé primaire à génération automatique de valeurs, de manière à éviter que la clé étrangère présente dans la table fille ne constitue une redondante affectant la performance de la base de données. Le changement de clé primaire dans la table mère conduit impérativement à un changement de clé étrangère dans toutes ses tables filles. Comme le montre la figure 2-42 où **Traitement** a été doté d'une nouvelle clé primaire **No type traitement**, un changement de nom et de type de la clé étrangère **Type traitement** (présente dans la table **Traitement patient**) a été effectué. L'attribut **Type traitement** doit tout de même être conservé dans la table **Traitement**.

FIGURE 2-42 **Modèle 2-41 où la clé primaire Type traitement est remplacée**



Cette forme d'optimisation doit être appliquée avec parcimonie. Le modèle de la figure 2-40 montre une table **Aéroport** dont la clé primaire simple est de type `String` mais dont la valeur est limitée à trois caractères alphabétiques, par exemple ***'YUL'***. Il n'y a pas lieu d'optimiser cette clé. En fait, si on évalue la pertinence des clés primaires simples de type `String` dans le modèle 2-40, aucune table ne devrait voir sa clé primaire simple changer à des fins d'optimisation.

Conséquences de l'application des techniques d'optimisation

Le modélisateur doit être conscient des conséquences de l'application des techniques énoncées sous les rubriques précédentes. Une conséquence immédiate est l'absence de correspondance entre le modèle conceptuel et le modèle relationnel dérivé optimisé.

Une autre conséquence est l'introduction d'incohérences dans le modèle à la suite d'un changement de clé primaire. Il faut se rappeler que si on change la clé primaire d'une table mère, les clés étrangères dans les tables filles doivent être changées pour assurer adéquatement l'association. Il incombe au modélisateur de modifier les clés étrangères des tables filles en changeant leur nom et en réduisant leur nombre lorsque la clé primaire composée de la table mère est remplacée par une clé primaire simple. S'il néglige de le faire, le modèle relationnel sera incohérent.

Une dernière conséquence est la *dénormalisation* du modèle relationnel car la fusion de tables pourrait conduire à la transgression de la troisième forme normale. Ce n'est pas en soi un problème si le modélisateur en accepte les effets secondaires : la table qui résulte d'une fusion peut comporter des lignes où plusieurs attributs sont laissés en blanc dans la base de données.

Dans le modèle 2-40, la table **Vol** possède des attributs tels que **Raison retard** ou **Raison annulation** qui seront en blanc pour tout vol n'ayant connu aucun retard et aucune annulation. Ce sont en conséquence des attributs pour lesquels la contrainte d'intégrité *{Non nul}* est totalement proscrite.

Validation du modèle relationnel de données

Avant de passer à la prochaine étape, soit la réalisation physique de la base de données découlant d'un modèle relationnel de données, il importe de valider ce dernier. Cette validation peut se mener en vérifiant trois caractéristiques essentielles qui assurent la cohérence du modèle :

1. Chaque table doit posséder une clé primaire ;
2. La plupart des associations sont de type *un à plusieurs* : multiplicité maximale de 1 d'une part et * d'autre part, soit une valeur numérique *strictement supérieure à un* ;
3. Chaque table fille, soit celle présente du côté de la multiplicité maximale *plusieurs* d'une association, doit être dotée d'une copie de la clé primaire ou secondaire de sa table mère avec mention {FK} ou {FK}{Cascade} ou {PK, FK}{Cascade}. Cette validation doit être effectuée avec grand soin car une même table peut être à la fois *table fille* pour plusieurs associations et *table mère* pour d'autres.
4. Exceptionnellement il peut exister quelques associations *un à un*. Dans un tel cas, une des deux tables doit être une table fille dotée d'une copie de la clé primaire ou secondaire de sa table mère avec mention {FK} ou {FK}{Cascade} ou {PK, FK}{Cascade}.

EXERCICES DE MODÉLISATION LOGIQUE DES DONNÉES

EXERCICE 2-1 Dériver le modèle relationnel de données (MRD) à partir du modèle conceptuel de données (MCD) donné comme solution de l'exercice 1-1. Optimiser le modèle considérant des multiplicités maximales 1 et 1 présentes sur une association. Valider le modèle final.

EXERCICE 2-2 Dériver le modèle relationnel de données à partir de la deuxième version du MCD donné comme solution de l'exercice 1-2. Il s'agit de la version ne comportant plus d'associations de degré supérieur. Procéder ensuite à l'optimisation du MRD en substituant à une clé primaire composée une clé primaire simple. Substituer dans les tables suivantes : **Fonction**, **Diplôme**, et **Institution d'enseignement** la clé primaire simple par une clé primaire à génération automatique de valeurs. Valider le modèle final.

EXERCICE 2-3 Dériver le modèle relationnel de données à partir de la deuxième version du MCD donné comme solution de l'exercice 1-3. Il s'agit de la version ne comportant plus d'associations de degré supérieur. Procéder ensuite à l'optimisation du MRD en substituant à une clé primaire composée une clé primaire simple. Valider le modèle final.

EXERCICE 2-4 Dériver le modèle relationnel de données à partir du MCD donné comme solution de l'exercice 1-4. Procéder ensuite à l'optimisation du MRD en substituant à une clé primaire composée une clé primaire simple. Débuter avec la table dérivée de l'entité **Opération**. Terminer avec la table dérivée de l'entité d'association **Outil utilisé**. Noter l'importance de procéder dans cet ordre. Valider le modèle final.

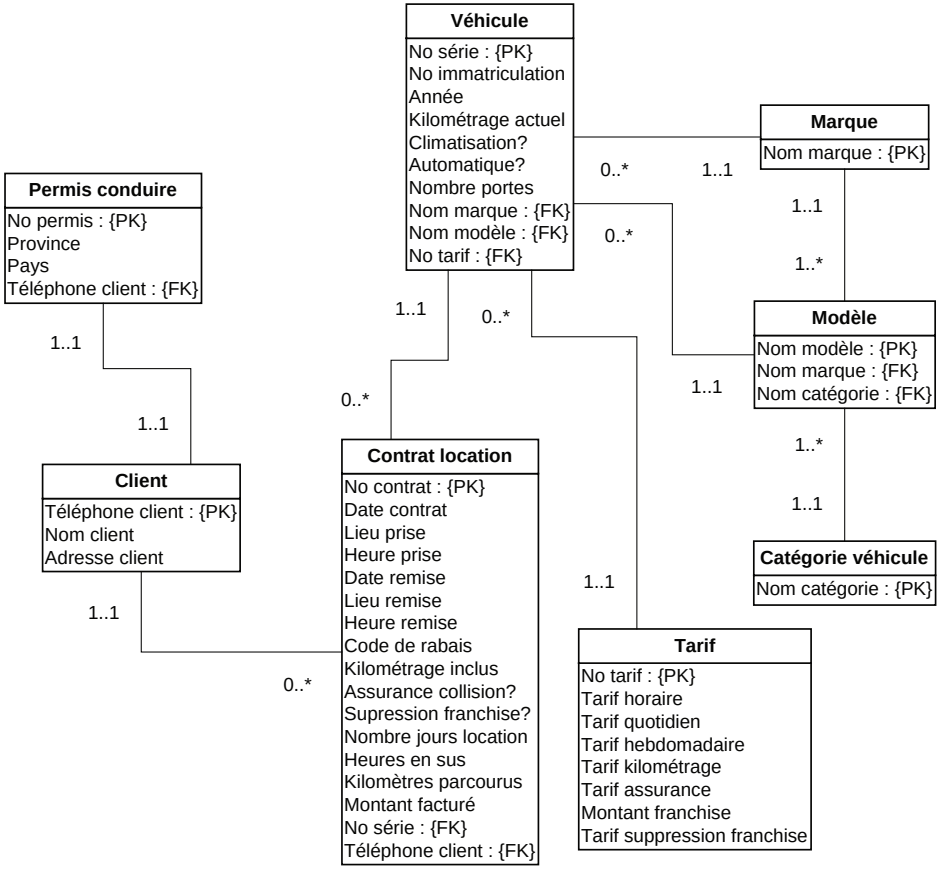
EXERCICE 2-5 Dériver le modèle relationnel de données à partir du MCD donné comme solution de l'exercice 1-5. Optimiser le modèle au fur et à mesure que les tables sont dérivées en donnant une clé primaire simple à génération automatique de valeurs à toutes les tables dérivées d'une association *plusieurs à plusieurs*. Valider le modèle final.

EXERCICE 2-6 Dériver le modèle relationnel de données à partir du MCD donné comme solution de l'exercice 1-6. Optimiser le modèle au fur et à mesure que les tables sont dérivées en donnant une clé primaire simple à génération automatique de valeurs à toutes les tables dérivées d'une association *plusieurs à plusieurs* ou d'une entité composant. Valider le modèle final.

SOLUTIONS DES EXERCICES DE MODÉLISATION LOGIQUE DES DONNÉES

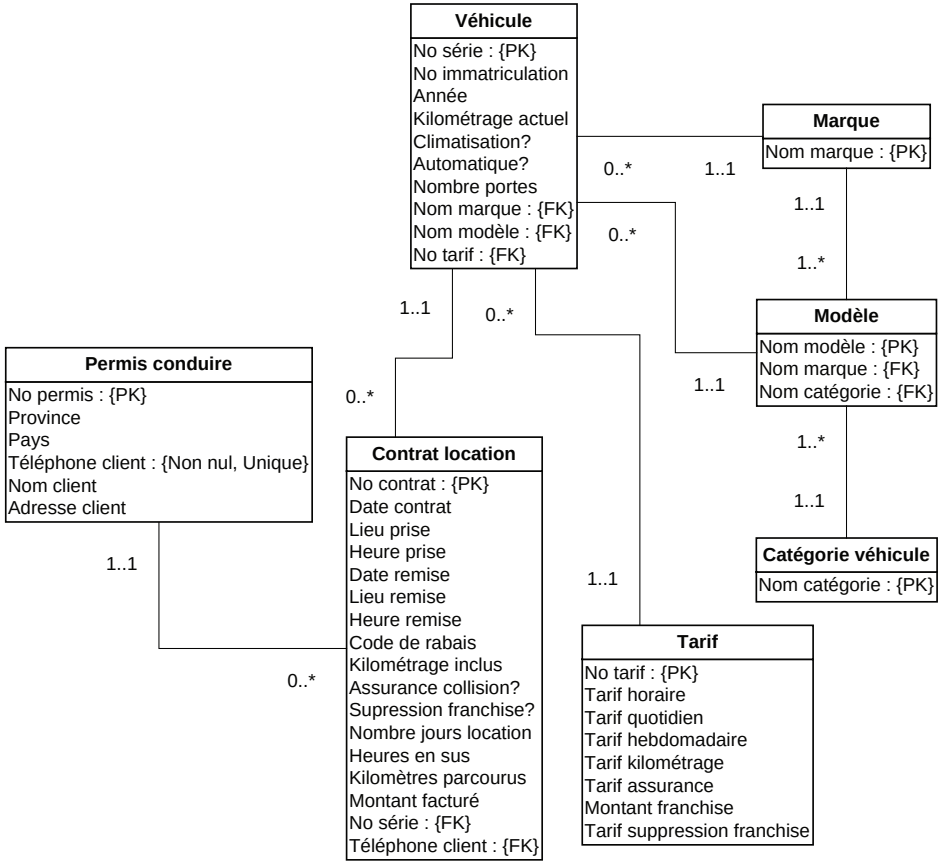
EXERCICE 2-1

Le modèle comporte une association *un à un*. Toutes les autres sont des associations *un à plusieurs*. Une première version du MRD avant optimisation est donnée ci-après.



Une association entre les tables **Permis conduire** et **Client** avec multiplicités 1..1 de part et d'autre peut donner lieu à la fusion des deux tables.

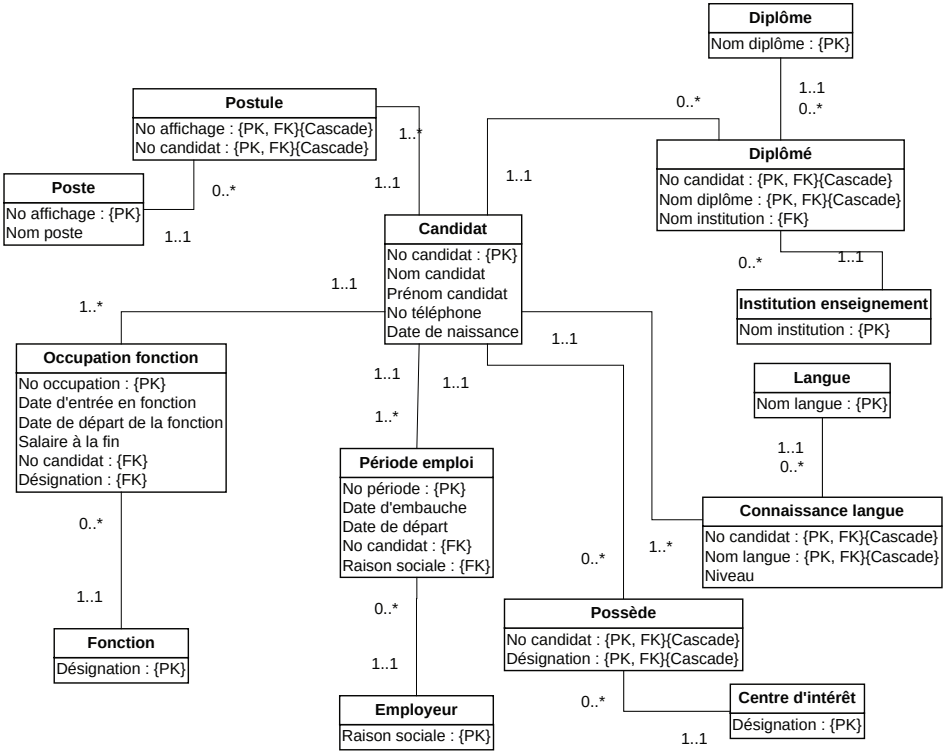
La table retenue pour la fusion est la table portant à titre de clé étrangère la clé primaire de l'autre table, soit **Permis conduire**. **Téléphone client** devient une clé secondaire pour cette table (mention {Non nul, Unique}). La table **Contrat location** est désormais associée à **Permis conduire**.


Véhicule
No série : {PK}
No immatriculation
Année
Kilométrage actuel
Climatisation?
Automatique?
Nombre portes
Nom marque : {FK}
Nom modèle : {FK}
No tarif : {FK}
Marque
Nom marque : {PK}
Modèle
Nom modèle : {PK}
Nom marque : {FK}
Nom catégorie : {FK}
Catégorie véhicule
Nom catégorie : {PK}
Permis conduire
No permis : {PK}
Province
Pays
Téléphone client : {Non nul, Unique}
Nom client
Adresse client
Contrat location
No contrat : {PK}
Date contrat
Lieu prise
Heure prise
Date remise
Lieu remise
Heure remise
Code de rabais
Kilométrage inclus
Assurance collision?
Supression franchise?
Nombre jours location
Heures en sus
Kilomètres parcourus
Montant facturé
No série : {FK}
Téléphone client : {FK}
Tarif
No tarif : {PK}
Tarif horaire
Tarif quotidien
Tarif hebdomadaire
Tarif kilométrage
Tarif assurance
Montant franchise
Tarif suppression franchise
0..*
1..1
1..1
1..*
0..*
1..1
1..1
0..*
0..*
1..1
1..*
1..1
1..1
0..*

EXERCICE 2-2

Les deux entités d'association **Diplômé** et **Connaissance langue** produisent chacune une table. Il en va de même pour les associations *plusieurs à plusieurs* **Postule** entre les entités **Poste** et **Candidat**, et **Possède** entre **Candidat** et **Centre d'intérêt**.

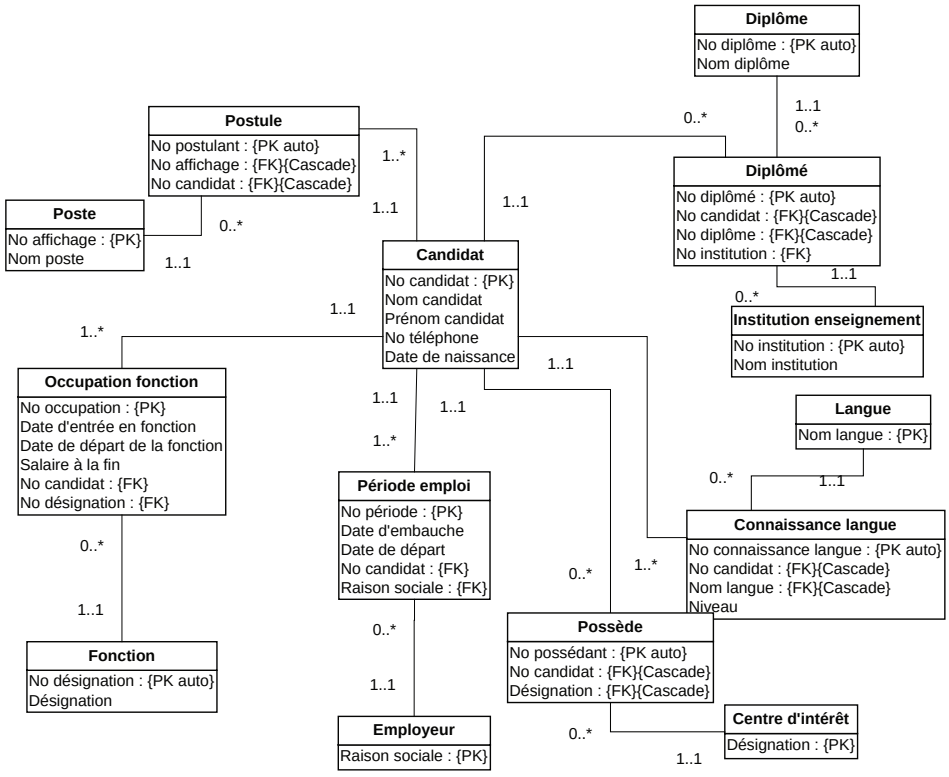
Le modèle relationnel suivant est la version non optimisée du MRD dérivé.



Le MRD ne comporte aucune association *un à un*, il n'y a donc pas lieu de procéder à la fusion de tables.

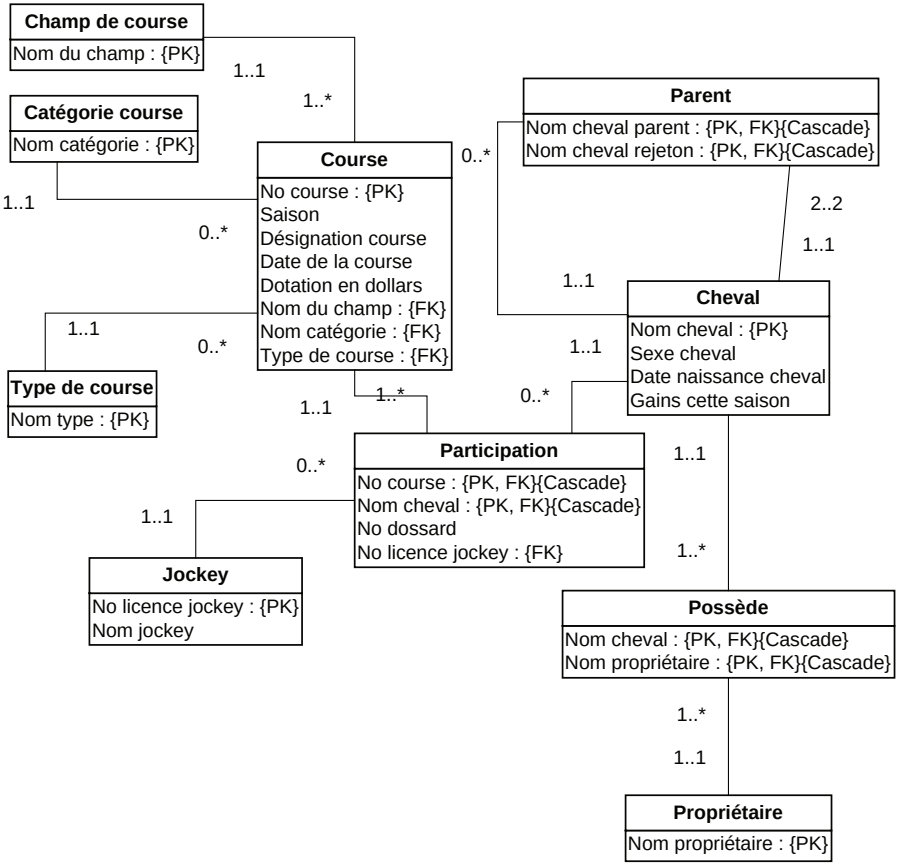
Les tables **Postule**, **Diplômé** et **Connaissance langue** peuvent se voir doter d'une clé primaire simple.

Compte tenu de la longueur de leur clé primaire simple de type `String`, il y a lieu de remplacer la clé primaire simple par une clé primaire à génération automatique de valeurs dans les tables suivantes : **Fonction**, **Diplôme**, et **Institution d'enseignement**. Ceci aura un impact sur le nom des clés étrangères présentes dans les tables associées **Occupation fonction** et **Diplômé**.

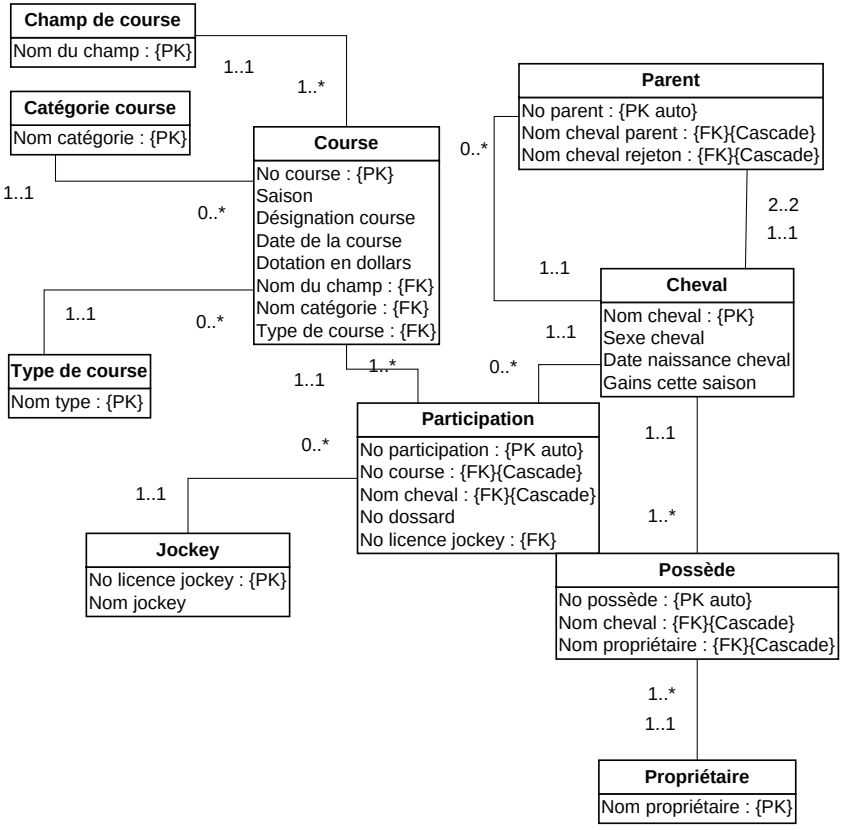

Diplôme
No diplôme : {PK auto}
Nom diplôme
Postule
No postulant : {PK auto}
No affichage : {FK}{Cascade}
No candidat : {FK}{Cascade}
Poste
No affichage : {PK}
Nom poste
Candidat
No candidat : {PK}
Nom candidat
Prénom candidat
No téléphone
Date de naissance
Diplômé
No diplômé : {PK auto}
No candidat : {FK}{Cascade}
No diplôme : {FK}{Cascade}
No institution : {FK}
Institution enseignement
No institution : {PK auto}
Nom institution
Occupation fonction
No occupation : {PK}
Date d'entrée en fonction
Date de départ de la fonction
Salaire à la fin
No candidat : {FK}
No désignation : {FK}
Période emploi
No période : {PK}
Date d'embauche
Date de départ
No candidat : {FK}
Raison sociale : {FK}
Langue
Nom langue : {PK}
Connaissance langue
No connaissance langue : {PK auto}
No candidat : {FK}{Cascade}
Nom langue : {FK}{Cascade}
Niveau
Possède
No possédant : {PK auto}
No candidat : {FK}{Cascade}
Désignation : {FK}{Cascade}
Fonction
No désignation : {PK auto}
Désignation
Employeur
Raison sociale : {PK}
Centre d'intérêt
Désignation : {PK}
1..1
0..*
1..*

EXERCICE 2-3

Le MCD comporte une association réflexive dont les multiplicités sont 2..2 d'une part et 0..* d'autre part. Une telle association est traitée comme une association binaire *plusieurs à plusieurs*. Une table en est dérivée, que nous appellerons **Parent**, comportant deux attributs **Nom cheval** avec des appellations différentes.

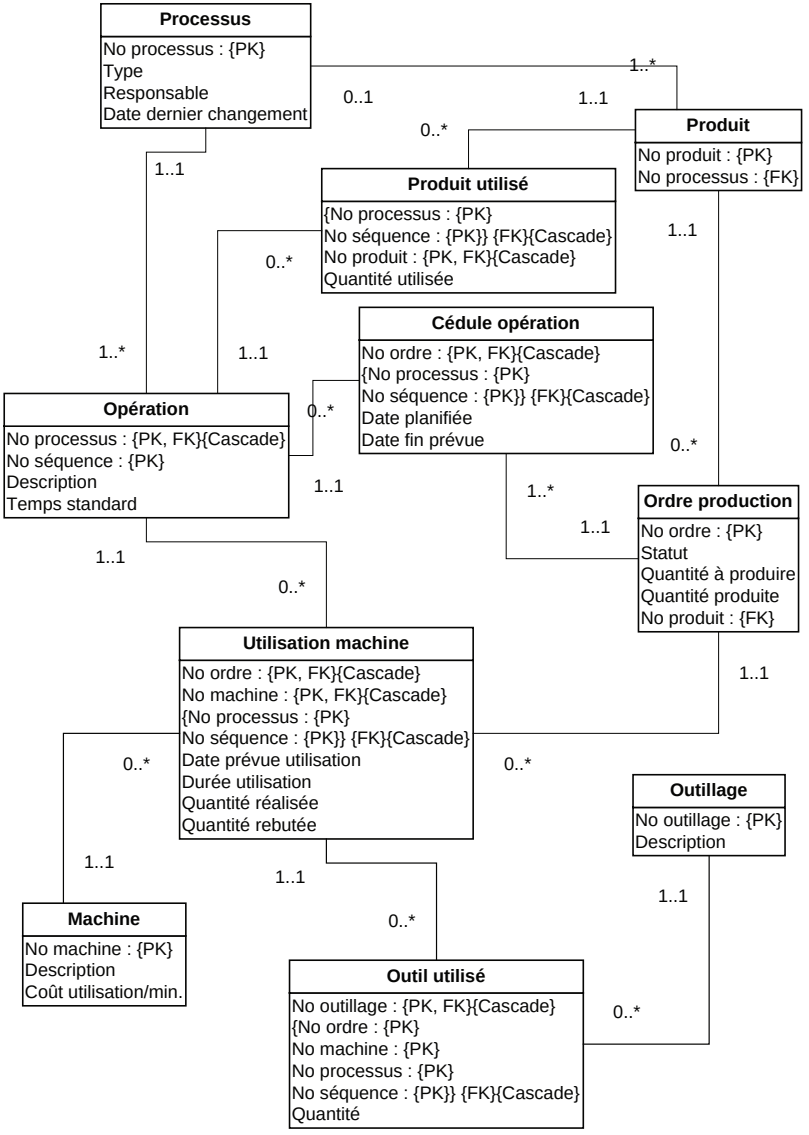


Après substitution de la clé primaire composée par une clé primaire simple à génération automatique de valeurs dans les tables **Parent**, **Participation** et **Possède**, le MRD optimisé ne comporte finalement que des tables avec une clé primaire simple.


Champ de course
Nom du champ : {PK}
Catégorie course
Nom catégorie : {PK}
Course
No course : {PK}
Saison
Désignation course
Date de la course
Dotation en dollars
Nom du champ : {FK}
Nom catégorie : {FK}
Type de course : {FK}
Parent
No parent : {PK auto}
Nom cheval parent : {FK}{Cascade}
Nom cheval rejeton : {FK}{Cascade}
Cheval
Nom cheval : {PK}
Sexe cheval
Date naissance cheval
Gains cette saison
Type de course
Nom type : {PK}
Participation
No participation : {PK auto}
No course : {FK}{Cascade}
Nom cheval : {FK}{Cascade}
No dossard
No licence jockey : {FK}
Jockey
No licence jockey : {PK}
Nom jockey
Possède
No possède : {PK auto}
Nom cheval : {FK}{Cascade}
Nom propriétaire : {FK}{Cascade}
Propriétaire
Nom propriétaire : {PK}
1..1
1..*
0..*
1..1
0..*
1..1
0..*
1..1
1..*
0..*
1..1
0..*
1..1
1..*
2..2
1..1
1..1
1..*
1..*
1..1

EXERCICE 2-4

La table **Opération** dérivée de l'association de composition est d'abord créée. Les entités d'association sont ensuite traitées en débutant par **Utilisation machine** dont la clé primaire est requise pour établir la clé primaire d'une autre entité d'association à laquelle elle est associée : **Outil utilisé**. Le MRD résultant et non optimisé donne lieu à de nombreuses tables dont la clé primaire est composée.

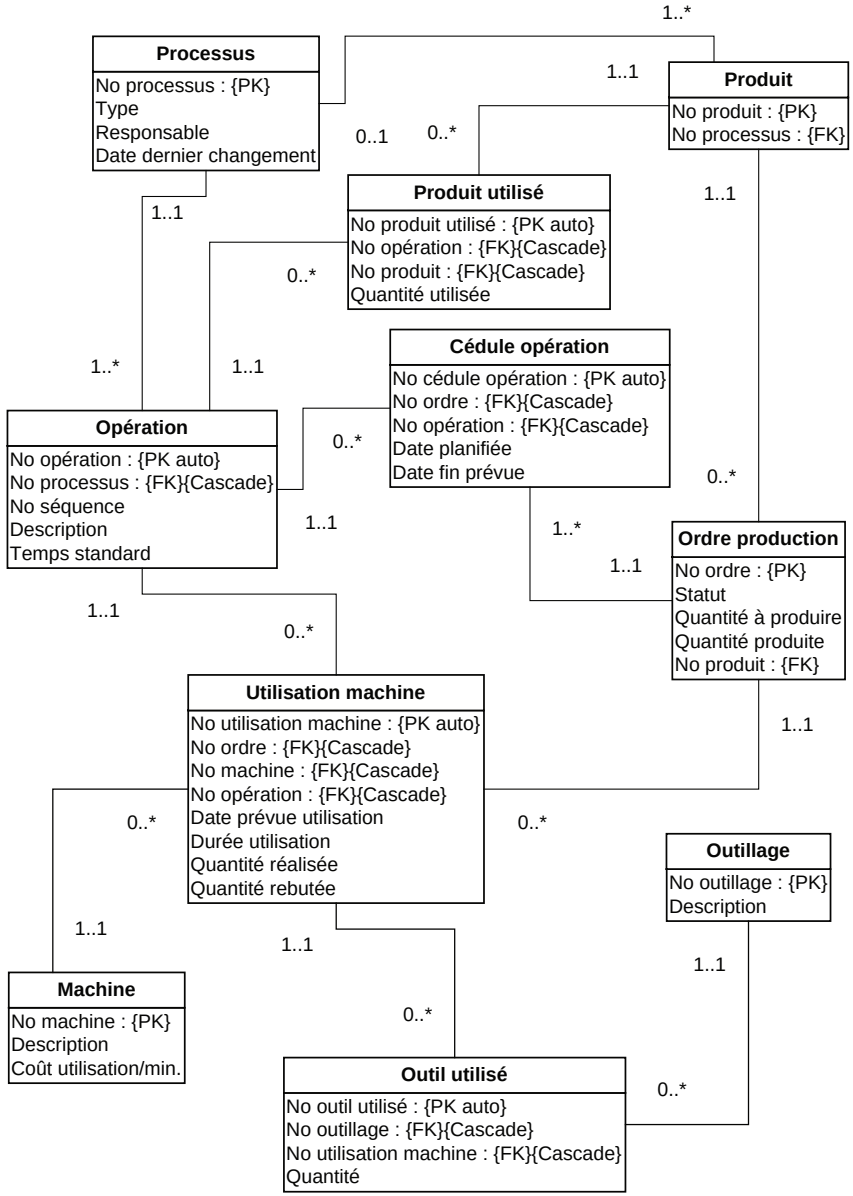


On comprend facilement toute l'importance d'optimiser les clés primaires composées en observant que la table **Outil utilisé** possède une clé primaire qui compte cinq attributs. De plus, bien que la substitution d'une clé composée par une clé primaire simple est en général pertinente dans les tables dérivées d'associations *plusieurs à plusieurs* et de degré supérieur, on constate dans cet exemple qu'elle l'est tout autant pour une table dérivée d'une entité composant. **Opération** est une table de cette sorte. Sa clé primaire combine son propre identifiant et l'identifiant de son composite et, comme **Opération** est impliquée dans deux associations avec entité d'association et une association de degré supérieur, sa clé primaire composée **(No processus, No séquence)** est reprise dans les trois tables dérivées à titre de clé secondaire composée. D'où l'importance d'entourer d'accolades les deux éléments de la clé étrangère qui font aussi partie de la clé primaire :

{No processus : {PK}
No séquence : {PK}} {FK}{CASCADE}

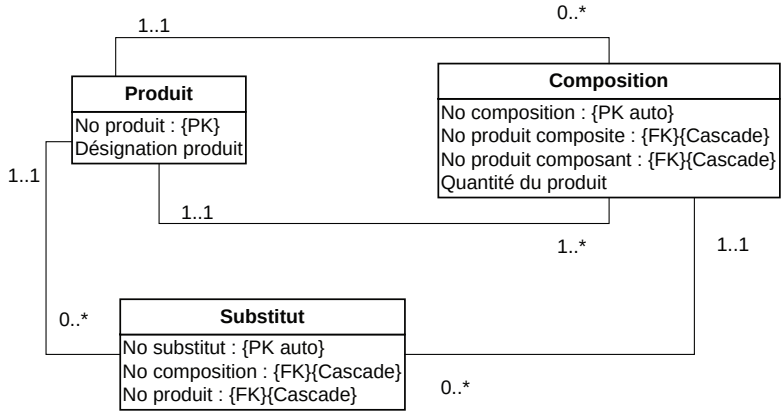
pour éviter que ces éléments ne soient individuellement considérés comme deux clés étrangères.

En remplaçant sa clé primaire composée par la clé primaire **No opération**, nous obtenons une optimisation en chaîne des tables dérivées des associations. Cela est particulièrement patent pour la table **Outil utilisé** qui ne comporte plus que deux clés étrangères au lieu des cinq présentes dans le modèle non optimisé. Le changement en chaîne des clés primaires, exigé afin que chaque clé étrangère soit révisée pour conduire au MRD final, est décrit ci-dessous.


Processus
No processus : {PK}
Type
Responsable
Date dernier changement
Produit
No produit : {PK}
No processus : {FK}
Produit utilisé
No produit utilisé : {PK auto}
No opération : {FK}{Cascade}
No produit : {FK}{Cascade}
Quantité utilisée
Cédule opération
No cédule opération : {PK auto}
No ordre : {FK}{Cascade}
No opération : {FK}{Cascade}
Date planifiée
Date fin prévue
Opération
No opération : {PK auto}
No processus : {FK}{Cascade}
No séquence
Description
Temps standard
Ordre production
No ordre : {PK}
Statut
Quantité à produire
Quantité produite
No produit : {FK}
Utilisation machine
No utilisation machine : {PK auto}
No ordre : {FK}{Cascade}
No machine : {FK}{Cascade}
No opération : {FK}{Cascade}
Date prévue utilisation
Durée utilisation
Quantité réalisée
Quantité rebutée
Outillage
No outillage : {PK}
Description
Machine
No machine : {PK}
Description
Coût utilisation/min.
Outil utilisé
No outil utilisé : {PK auto}
No outillage : {FK}{Cascade}
No utilisation machine : {FK}{Cascade}
Quantité
1..*
1..1
0..1
0..*
1..1
1..1
0..*
1..*
1..1
0..*
1..1
1..*
1..1
0..*
1..1
0..*
0..*
1..1
1..1
0..*
1..1
0..*

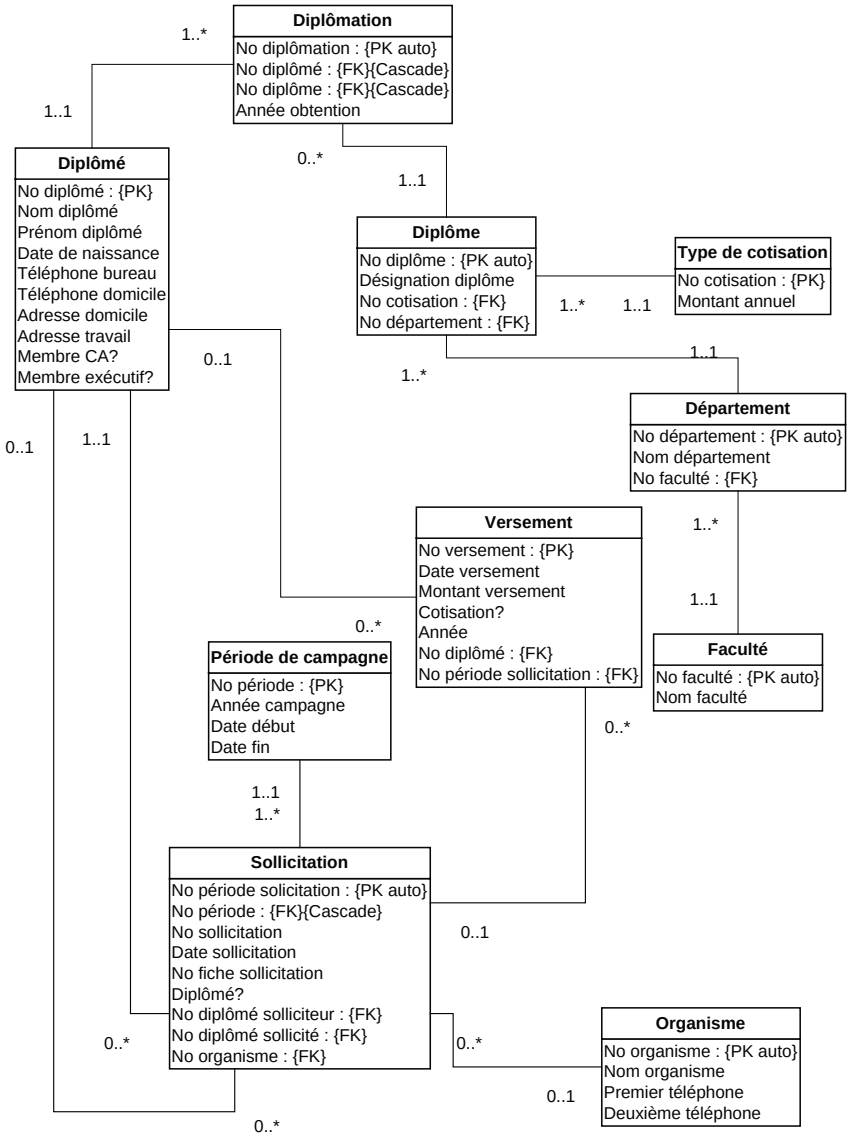
EXERCICE 2-5

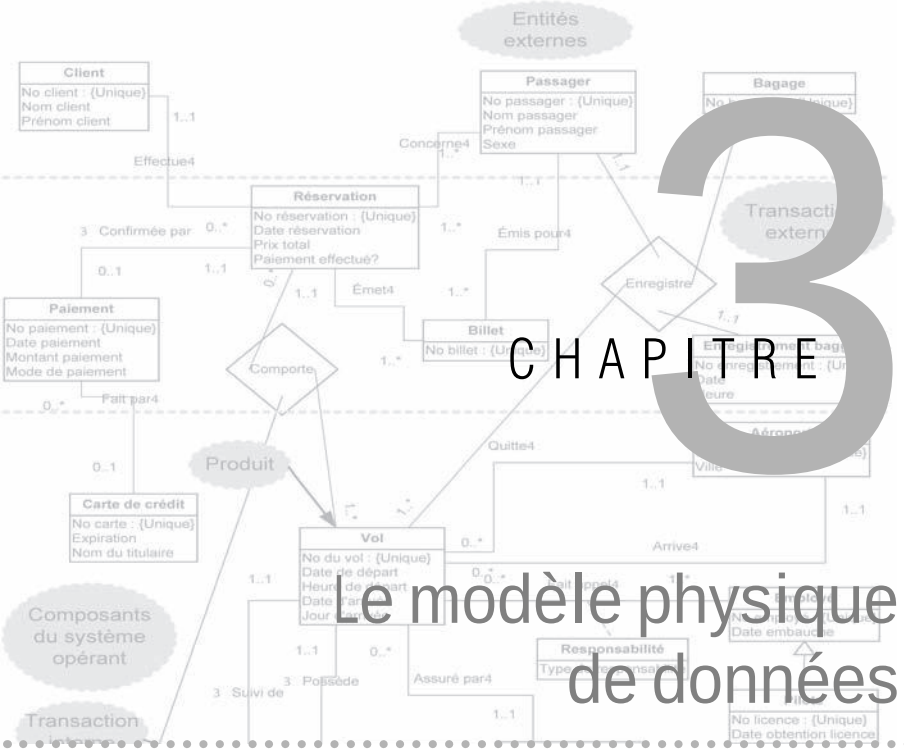
Le MRD suivant est pleinement optimisé. Notons que le choix du nom de la table **Substitut** est plus approprié que **Possède un substitut**, le nom donné à l'association *plusieurs à plusieurs du* MCD.



EXERCICE 2-6

Toutes les tables comportant avant optimisation une clé primaire simple de type `String` ont été dotées d'une clé primaire simple à génération automatique de valeurs. Toutes les tables dérivées d'une composition ou d'une association *plusieurs à plusieurs* possèdent par ailleurs une clé primaire simple de cette nature. Notons que la table **Sollicitation** dispose de deux clés étrangères l'associant à la table **Diplômé** auxquelles on a pris soin de donner des noms différents.





CHAPITRE 3

Le modèle physique de données

OBJECTIFS

- Versions des instructions de définition de données du langage SQL, notamment celles répondant à la norme ANSI 92.
- Règles et principes assurant la conversion d'un modèle relationnel en un modèle physique de données exprimé dans le langage SQL.
- Détail de la formulation d'un modèle physique de données par le biais d'un script SQL conforme aux exigences du SGBD MS Access.
- Comment réaliser un modèle physique de données avec le SGBD MS Access sans faire appel à un script SQL.

Ce chapitre traite de la dernière étape dans la réalisation d'une base de données soit l'élaboration d'un *modèle physique de données*. Nous le consacrons essentiellement aux techniques de réalisation d'un modèle physique de données avec MS Access, à partir d'un modèle relationnel.

Modèle physique de données ▸ Un modèle de données découlant d'un modèle logique qui spécifie les détails d'implantation du modèle logique dans un SGBD, habituellement formulés grâce à un **script de définition de données** (*Physical Data model*).

Script de définition de données ▸ Une suite d'instructions de définition de données rédigées selon les exigences du langage SQL. Ces instructions spécifient le **schéma physique** de la BD notamment la structure des tables ainsi que toutes les contraintes d'intégrité applicables aux données stockées dans les tables (*Data Definition Script*).

Avant d'aborder la question de la transition du modèle relationnel au modèle physique de données, étudions les possibilités offertes par le langage SQL pour la définition de données.

SQL COMME LANGAGE DE DÉFINITION DE DONNÉES

Nous avons vu au chapitre 1 que SQL est bien plus qu'un langage d'interrogation de bases de données. Il permet au concepteur d'une base de données de créer les tables et les associations entre ces tables ainsi que de formuler des contraintes d'intégrité sur les attributs et les associations. Les organismes de normalisation américain ANSI (*American National Standards Institute)* et international ISO (*International Organization for Standardization*) ont adopté une norme commune pour ce langage. Elle est connue sous le nom ANSI-92 [ANS 92] ou SQL2 ISO [ISO 92]. Des normes plus récentes existent dont SQL3 ISO versions 1999 et 2003.

Les SGBD relationnels actuellement sur le marché, dont MS Access 2002 et ses versions postérieures, se conforment généralement à la norme ANSI-92, tout au moins au premier niveau de la norme. C'est la raison pour laquelle tout au long de ce chapitre nous adoptons cette norme pour la formulation des modèles physiques. La norme ANSI-92 permet d'établir le type de données d'un attribut conforme au type attribué aux niveaux conceptuel et relationnel.

Le tableau 3-1 présente la correspondance entre le type de données au niveau logique et son équivalent au niveau physique en vertu de la norme ANSI-92. Une valeur entière au niveau logique peut être implémentée comme un *entier* (**`SMALLINT`**) ou un *entier long* (**`INTEGER`**). En Access le type *entier* permet de stocker des valeurs entières entre –32 768 et 32 767 alors que *l'entier long* stocke des valeurs entières entre –2 147 483 648 et 2 147 483 647. Il en va de même pour les nombres réels qui peuvent être du type *réel simple* (**`FLOAT`**) ou *réel double précision* (**`DOUBLE PRECISION`**). Access stocke dans un

attribut *réel simple* des nombres compris entre –3,402823E38 et –1,401298E–45 pour les valeurs négatives et entre 1,401298E–45 et 3,402823E38 pour les valeurs positives. Quant au type *réel double précision* il stocke des nombres compris entre –1,79769313486231E308 et –4,94065645841247E–324 pour les valeurs négatives et entre 4,94065645841247E–324 et 1,79769313486231E308 pour les valeurs positives. Le type **VARCHAR**, appelé type *Texte* en Access, stocke quant à lui des chaînes de caractères de longueur variant entre 0 et 255 caractères.

TABLEAU 3-1 **Types de données des attributs au niveau logique et au niveau physique**

Type au niveau logique	Signification	Type ANSI-92
`Integer`	Valeur numérique entière	**SMALLINT** / **INTEGER**
`Real`	Valeur numérique réelle	**FLOAT** / **DOUBLE PRECISION**
`String`	Chaîne de caractères	**VARCHAR**
`Date`	Date du calendrier	**DATE**
`Boolean`	Valeur Vrai ou Faux	**BIT**
`enum{}`	Liste de valeurs admissibles	Clause **CHECK**

Le type énuméré n'a pas son équivalent direct dans la norme ANSI-92 et, comme toutes les autres contraintes de domaine, il est réalisé avec la clause SQL **CHECK** que nous verrons sous peu.

MS Access dispose d'autres types de données physiques non présents dans la norme ANSI-92. Il s'agit des types de données appelés *Monnaie*, *Numéroauto* et *Mémo* dans la livrée française du logiciel. Le type *Numéroauto*, **IDENTITY** en SQL, est le moyen dont on dispose en Access pour créer une clé primaire simple à génération automatique de valeurs. Dans sa version SQL, le type **IDENTITY** est doté de deux paramètres, la valeur entière initiale générée et l'incrément utilisé pour les générations subséquentes. Par exemple, le type **IDENTITY(10,5)** sera utilisé pour générer des valeurs pour la clé primaire dans la séquence : 10, 15, 20, 25, etc. Si **IDENTITY** ne dispose d'aucun paramètre, la séquence débute à 1 et est incrémentée de 1 à chaque génération.

Le type physique **MEMO** permet de passer outre à la limite de 255 imposée à la longueur d'une chaîne de caractères de type **VARCHAR**.

TABLEAU 3-2 **Autres types de données au niveau physique**

Type au niveau logique	Signification	Type physique non ANSI-92
	Valeur numérique réelle portant symbole monétaire	**`CURRENCY`**
Entier long sans doublon, mention {PK auto} au niveau logique	Valeur numérique entière sans doublon générée automatiquement débutant à *d* et incrémenté de *i* à chaque fois	**`IDENTITY(d,i)`**
	Chaîne de caractères de plus de 255 caractères, mais moins de 65 535	**`MEMO`**

Création de tables avec SQL

L'instruction qui permet de créer une table dans une BD avec le langage SQL est appelée **`CREATE TABLE`**. Cette instruction peut être relativement complexe car elle doit spécifier :

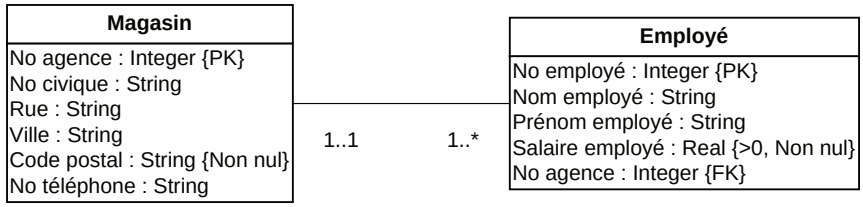
- Le nom de la table
- Le nom de chaque attribut (colonne) de la table
- Le domaine de la colonne : type de données physique et contraintes d'intégrité sémantiques (unique, non nul, valeurs acceptables, etc.)
- La valeur par défaut de la colonne
- La clé primaire de la table et ses clés secondaires
- Les clés étrangères de la table et leur pendant : les clés des tables associées
- Les contraintes d'intégrité référentielles

Avant de présenter la syntaxe formelle de l'instruction **`CREATE TABLE`**, considérons quelques exemples tirés du chapitre 2.

Table avec clé primaire simple

Le modèle qui suit ne comporte que des tables avec clé primaire simple et une association *un à plusieurs*. Il représente la forme la plus courante d'association dans un modèle relationnel.

FIGURE 3-1 **Modèle relationnel comportant deux tables liées par une association *un à plusieurs***



Dans le script SQL décrivant le modèle physique, une table mère est toujours créée avant ses tables filles. Ainsi la table **Magasin** devra d'abord être créée grâce à l'instruction suivante :

```
CREATE TABLE Magasin
([No agence] SMALLINT NOT NULL PRIMARY KEY,
[No civique] VARCHAR(10),
[Rue] VARCHAR(30),
[Ville] VARCHAR(30),
[Code postal] VARCHAR(6) NOT NULL,
[No téléphone] VARCHAR(10));
```

Le nom des colonnes est systématiquement mis entre crochets pour tolérer l'espace séparateur entre les mots d'un nom composé. Chaque colonne dispose d'un type de données physique en correspondance avec le type de données du modèle relationnel. Le nom de la clé primaire est suivi des clauses **NOT NULL** et **PRIMARY KEY**. Une clé primaire doit en effet répondre à la contrainte d'intégrité d'entité : sa valeur ne peut être nulle (clause **NOT NULL**) et il ne peut y avoir de doublons, soit la même valeur donnée à la clé primaire sur plusieurs lignes de la table (clause **PRIMARY KEY**).

L'attribut **Code postal** possède une contrainte d'intégrité stipulant que sa valeur ne peut être nulle, ce qui se traduit par la clause **NOT NULL** placée à la suite de son type de données **VARCHAR(6)**.

Le script nécessaire à la création de la table **Employé** est donné ci-après.

```
CREATE TABLE Employé
([No employé] SMALLINT NOT NULL PRIMARY KEY,
[Nom employé] VARCHAR(20),
[Prénom employé] VARCHAR(20),
[Salaire employé] CURRENCY NOT NULL,
[No agence] SMALLINT NOT NULL
 REFERENCES Magasin([No agence]),

CONSTRAINT SalaireSupÀZéro CHECK([Salaire employé]>0))
```

Cette fois-ci la table possède une clé étrangère et elle doit être spécifiée avec la clause **REFERENCES**. **No agence** est un entier dont la valeur ne peut être nulle car la multiplicité minimale est 1 du côté de la table mère **Magasin** d'où la présence d'une clause **NOT NULL** qui suit le type de données **SMALLINT**. La clause **REFERENCES** stipule qu'il s'agit d'une clé étrangère associée à la table **Magasin** par le biais de sa clé primaire **No agence**:

```
REFERENCES Magasin([No agence])
```

La clause **REFERENCES** représente une contrainte d'intégrité référentielle en ajout. Elle signifie que, lors de l'ajout d'une ligne dans la table fille **Employé**, le **No agence** de l'employé doit être présent dans la table mère **Magasin**. De plus, si une ligne de la table **Magasin** est supprimée, elle ne pourra l'être que si aucune ligne ne lui est associée dans la table **Employé**. Aucune action ne sera entreprise visant notamment à effacer automatiquement toutes les lignes associées dans **Employé**.

La dernière clause de l'instruction permettant de créer la table **Employé** est une contrainte de domaine portant sur l'attribut **Salaire employé** indiquant qu'il doit en tout temps être strictement supérieur à 0:

```
CONSTRAINT SalaireSupÀZéro CHECK([Salaire employé]>0)
```

Chaque contrainte portant sur des colonnes de la table doit faire appel à la clause **CONSTRAINT** pour lui accorder un nom (ici **SalaireSupÀZéro**) et formuler ensuite une condition de validité avec la clause **CHECK**. Cette dernière clause permet notamment de définir des conditions de validité, comme on le ferait grâce à la propriété *Valide si* en «Mode création de table» dans MS Access.

Le script de création de la table **Employé** donné plus haut, bien qu'il permette d'assurer la multiplicité minimale 1 du côté de la table mère **Magasin** par la présence de la clause **NOT NULL** sur l'attribut **No agence**, n'implante pas la multiplicité minimale 1 du côté de **Employé**.

Pour ce faire, il faudrait ajouter une nouvelle contrainte au script de création de la table **Magasin**, une contrainte qui permet de vérifier qu'un magasin emploie au moins un employé. L'instruction **ALTER TABLE** permet un tel ajout sur une table existante.

```
ALTER TABLE Magasin
ADD CONSTRAINT AuMoinsUnEmployé
CHECK (NOT EXISTS(SELECT COUNT(Employé.[No employé]) AS
[CompteEmployéDuMagasin], Magasin.[No agence] FROM
Magasin LEFT JOIN Employé ON Magasin.[No agence] =
Employé.[No agence]
GROUP BY Magasin.[No agence]
HAVING (((Count(Employé.[No employé]))=0))))
```

La contrainte doit être ajoutée à la table **Magasin** après la création de la table **Employé** puisque, comme le montre le script, elle fait référence aux colonnes de **Employé**. La contrainte **AuMoinsUnEmployé**, relativement complexe et correcte sur le plan syntaxique, ne peut malheureusement pas être mise en œuvre car elle est en contradiction avec la contrainte d'intégrité référentielle déjà appliquée à **No agence**.

Voici pourquoi. Rappelons que la contrainte d'intégrité référentielle stipule qu'un nouvel employé ne peut être ajouté dans la table que si le **No agence** l'est déjà dans la table **Magasin**. Les données du magasin doivent donc exister dans la BD avant de pouvoir ajouter les données des employés qui y travaillent. En conséquence, lorsque les données du magasin sont entrées dans la table **Magasin**, aucun employé de ce magasin n'existe dans la table **Employé**, ce qui transgresse la contrainte de multiplicité minimale 1 qui veut qu'un magasin emploie au moins un employé. C'est un cercle vicieux. La présence d'une contrainte a pour effet de rendre l'autre contrainte inapplicable. C'est un problème que l'on appelle *interbloquage* en langage technique.

Cette situation se rencontre lorsqu'il y a multiplicité maximale 1 du côté de la table fille et qu'une contrainte d'intégrité référentielle s'applique à la clé étrangère assurant cette association. Pour éviter tout conflit, une des contraintes ne sera pas réalisée à travers le script de création des tables. La contrainte assurant une multiplicité maximale 1 du côté de la table fille sera mise en œuvre de façon distincte. On choisira plutôt de miser sur des

instructions de programmation pour vérifier si la BD respecte ce type de contrainte tout comme c'est le cas des contraintes inter-associations par ailleurs. Le thème de la programmation des contraintes à l'aide d'un langage de programmation n'est pas couvert dans cet ouvrage.

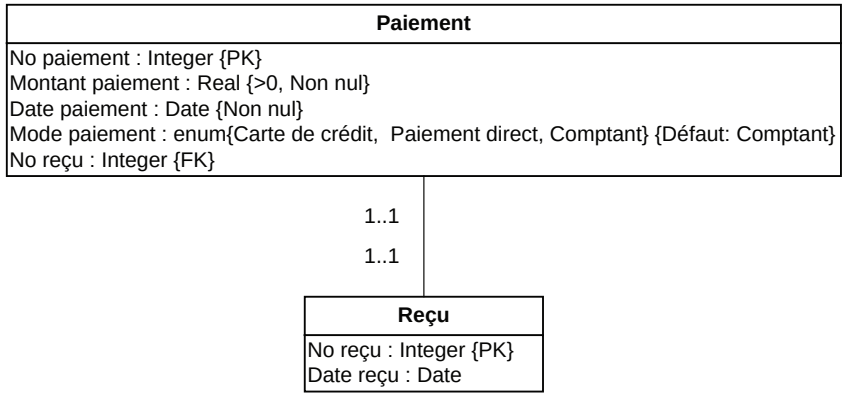
Le tableau 3-3 se veut un synopsis des exigences du modèle relationnel de la figure 3-1 et la formulation de ces exigences dans le modèle physique à l'aide d'un script de création de tables.

TABLEAU 3-3 **Résumé des clauses du script de création de tables requises pour réaliser le modèle physique correspondant au modèle 3-1**

Exigence	Niveau logique	Signification	Niveau physique : Clause requise dans le script
1.	{PK}	Cet attribut constitue la clé primaire : il ne peut avoir une valeur nulle et avoir des doublons (intégrité d'entité)	**`NOT NULL PRIMARY KEY`**
2.	{Non nul}	La valeur de cet attribut ne peut être nulle (laissée en blanc)	**`NOT NULL`**
3.	{FK}	Cet attribut constitue une clé étrangère avec intégrité référentielle en ajout	**`REFERENCES`** *`Table_mère(Clé_mère)`* Le type de données de la clé primaire doit être le même que celui de la clé étrangère
4.	Multiplicité *minimale* 1 côté **mère**	La clé étrangère comporte obligatoirement une valeur	**`NOT NULL`** pour la clé étrangère qui réalise l'association
5.	Multiplicité *minimale* 1 côté **fille**	On doit avoir au moins une ligne de la table fille associée à une ligne de la table mère	Ne pas mettre en œuvre avec **`CREATE TABLE`** car il y a un conflit avec l'intégrité référentielle
6.	Multiplicité *maximale* 1 côté **mère**	Une ligne fille ne peut être associée au plus qu'à une ligne mère	Aucune clause requise car dans le modèle relationnel une clé étrangère ne peut avoir qu'une seule valeur à la fois
7.	{>*Nombre*}	Cet attribut possède une contrainte de domaine	**`CONSTRAINT`** *`Nom_contrainte`* **`CHECK`***`(Nom_attribut>Nombre)`*

La figure 3-2 montre cette fois un modèle relationnel avec association *un à un* qui pourrait être optimisé mais ne l'a pas été. Cet exemple permet de présenter un script de création des tables qui reflète sur plusieurs aspects les mêmes exigences que pour le modèle 3-1 en plus de répondre à certaines exigences originales.

FIGURE 3-2 **Modèle relationnel comportant deux tables liées par une association *un à un***



La table **Reçu**, qui est la table mère pour l'association, est la première créée.

```
CREATE TABLE Reçu
([No reçu] SMALLINT NOT NULL PRIMARY KEY,
[Date reçu] DATE)
```

Sa table fille **Paiement** est créée avec le script suivant :

```
CREATE TABLE Paiement
([No paiement] SMALLINT NOT NULL PRIMARY KEY,
[Montant paiement] CURRENCY NOT NULL,
[Date paiement] DATE NOT NULL,
[Mode paiement] VARCHAR(20) NOT NULL DEFAULT Comptant,
[No reçu] SMALLINT NOT NULL UNIQUE
 REFERENCES Reçu([No reçu]),

CONSTRAINT MontantSupÀZéro CHECK([Montant paiement]>0),

CONSTRAINT ModePaiementValide CHECK([Mode paiement]
IN ('Carte de crédit', 'Carte de débit', 'Comptant')))
```

Les clauses **NOT NULL** inscrites à la suite des attributs **Montant paiement**, **Date paiement** et **Mode paiement** reflètent les contraintes exprimées dans le modèle relationnel pour ces attributs.

TABLEAU 3-4 **Résumé des clauses du script de création de tables requises pour réaliser le modèle physique correspondant au modèle 3-2**

Exigence	Niveau logique	Signification	Niveau physique : Clause requise dans le script
1.	{PK}	Cet attribut constitue la clé primaire : il ne peut avoir une valeur nulle et avoir des doublons (intégrité d'entité)	**NOT NULL PRIMARY KEY**
2.	{Non nul}	La valeur de cet attribut ne peut être nulle (laissée en blanc)	**NOT NULL**
3.	{FK}	Cet attribut constitue une clé étrangère avec intégrité référentielle en ajout	**REFERENCES** *Table_mère*(*Clé_mère*) Le type de données de la clé primaire doit être le même que celui de la clé étrangère
4.	Multiplicité *minimale* 1 côté **mère**	La clé étrangère comporte obligatoirement une valeur	**NOT NULL** pour la clé étrangère qui réalise l'association
5.	Multiplicité *minimale* 1 côté **fille**	On doit avoir au moins une ligne de la table fille associée à une ligne de la table mère	Ne pas mettre en œuvre avec **CREATE TABLE** car il y a un conflit avec l'intégrité référentielle
6.	Multiplicité *maximale* 1 côté **mère**	Une ligne fille ne peut être associée au plus qu'à une ligne mère	Aucune clause requise car dans le modèle relationnel un attribut, donc une clé étrangère, ne peut avoir qu'une seule valeur à la fois
7.	{>*Nombre*}	Cet attribut possède une contrainte de domaine	**CONSTRAINT** *Nom_contrainte* **CHECK(***Nom_attribut***>***Nombre***)**
8.	{Défaut *valeur*}	Cet attribut dispose d'une valeur par défaut	**DEFAULT** *Valeur_par_défaut*
9.	Multiplicité *maximale* 1 côté **fille**	La clé étrangère ne peut avoir des doublons	**UNIQUE**
10.	enum{val_1, val_2, …, val_n }	Cet attribut possède une contrainte de domaine	**CONSTRAINT** *Nom_contrainte* **CHECK(***Nom_attribut* **IN (**val_1, val_2, …, val_n**))**

L'attribut **Mode paiement** possède une valeur par défaut, la chaîne de caractères **Comptant**, en conformité avec les exigences du modèle relationnel.

La clause **NOT NULL** qui suit le type de données **SMALLINT** de la clé étrangère **No reçu** exprime sur le plan physique l'exigence de multiplicité minimale 1 du côté de la table fille **Paiement** comme dans le modèle 3-1 où le script de création de la table **Employé** spécifie une clé étrangère **No agence** qui porte aussi la **NOT NULL** pour implanter l'exigence de multiplicité minimale 1 du côté de la table fille **Employé**.

On note de plus la présence de la clause **UNIQUE** qui assure ici l'implémentation de la multiplicité maximale 1 du côté le la table mère **Reçu**. En effet, il ne peut y avoir deux lignes de la table **Paiement** ayant la même valeur pour **No reçu**.

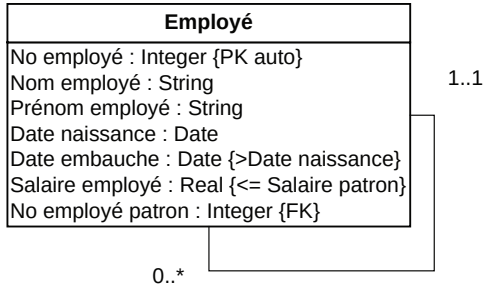
La contrainte **MontantSupÀZéro** restreint le domaine de **Montant paiement** à des valeurs strictement supérieures à zéro. La contrainte **ModePaiementValide** pour sa part spécifie un domaine pour l'attribut **Mode paiement** qui est limité aux trois chaînes de caractères: **'Carte de crédit'**, **'Carte de débit'** et **'Comptant'**.

La figure 3-3 illustre une situation où la table a une clé primaire simple mais cette fois elle est associée à elle-même. L'association réflexive est nécessaire pour lier une ligne comportant les données sur un employé à une autre ligne de la même table comportant les données sur son patron.

Une contrainte bien banale exige que la date d'embauche soit supérieure à la date de naissance. Une autre contrainte, plus singulière, stipule que le salaire de l'employé doit être inférieur ou égal à celui de son patron.

La table **Employé** est à la fois mère et fille, sa clé primaire est à génération automatique de valeurs.

FIGURE 3-3 **Modèle relationnel comportant une table liée à elle-même par une association *un à plusieurs***



Le modèle physique doit fait appel à la clause **IDENTITY** comme type de données de **No employé** car on exige une génération automatique de valeurs. Le type de l'attribut, comme pour toutes les clés primaires, doit être suivi des clauses **NOT NULL** et **PRIMARY KEY**. Soulignons qu'il s'agit d'un attribut dont le type est techniquement un *entier long*.

Le script doit formuler deux contraintes. Une première concerne les valeurs admissibles pour l'attribut **Date embauche**. Sa valeur est en relation avec un attribut de la même table dont la valeur est disponible sur la même ligne. Il suffit de mettre en relation les deux attributs par le biais de la clause **CONSTRAINT** suivante :

```
CONSTRAINT EmbaucheApresNaissance CHECK([Date embauche]>
[Date naissance])
```

La deuxième contrainte est plus complexe à implanter. Le modèle relationnel montre qu'une ligne de **Employé** est toujours associée à une et une seule ligne de la même table mais cette ligne ne peut être la même car en général un employé n'est pas son propre patron, sauf pour un seul employé. Si on exclut le cas de l'employé qui est son propre patron, le salaire du patron n'est pas donné sur la même ligne que les autres données concernant l'employé. Une opération de jointure réalisée sur la même table avec la clause **SELECT** doit permettre d'associer le salaire d'un employé avec le salaire de son patron et de vérifier ensuite qu'il n'existe aucune situation où le salaire d'un patron est inférieur à celui d'un de ses employés. La vérification peut être faite par une contrainte exprimée en ces termes :

```
CONSTRAINT SalaireInfEgalAuPatron CHECK(NOT EXISTS
(SELECT Employé.[Salaire employé]
FROM Employé INNER JOIN Employé AS Employé_1 ON
Employé.[No employé]=Employé_1.[No employé patron]
WHERE (((Employé.[Salaire employé])<Employé_1.[Salaire
employé])))
```

La table **Employé** est ainsi créée comme s'il s'agissait d'une table mère sur laquelle les deux contraintes sont implantées.

```
CREATE TABLE Employé
([No employé] IDENTITY NOT NULL PRIMARY KEY,
[Nom employé] VARCHAR(20),
[Prénom employé] VARCHAR(20),
[Date naissance] DATE,
[Date embauche] DATE,
[Salaire employé] CURRENCY,
[No employé patron] INTEGER NOT NULL
 REFERENCES Employé([No employé]),
```

```
CONSTRAINT EmbaucheApresNaissance CHECK([Date embauche]>
[Date naissance]),

CONSTRAINT SalaireInfEgalAuPatron CHECK(NOT EXISTS
(SELECT Employé.[Salaire employé]
FROM Employé INNER JOIN Employé AS Employé_1 ON
Employé.[No employé]=Employé_1.[No employé patron]
WHERE (((Employé.[Salaire employé])<Employé_1.[Salaire
employé])))))
```

Il importe de souligner que la clé étrangère **No employé patron** doit être de type **INTEGER** car le type de données **IDENTITY** de la clé primaire associée représente un *entier long*. L'utilisation du type **SMALLINT** (un entier *court*) aurait conduit à une incohérence de type. Le tableau 3-5 résume les exigences

TABLEAU 3-5 **Résumé des clauses du script de création de tables requises pour réaliser le modèle physique correspondant au modèle 3-3**

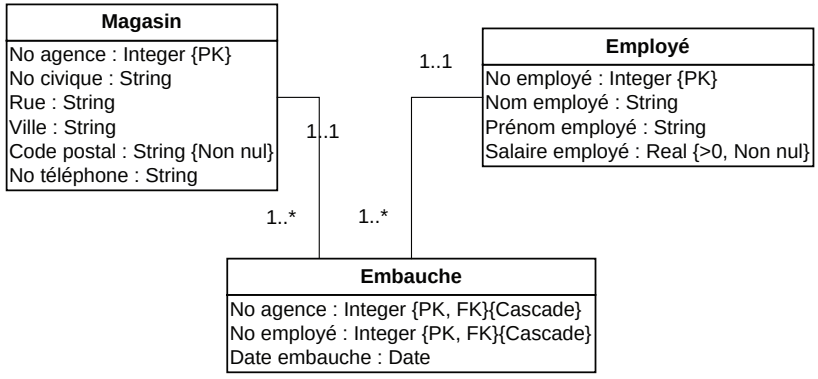
Exigence	Niveau logique	Signification	Niveau physique : Clause requise dans le script
11.	{PK auto}	Cet attribut constitue une clé primaire avec génération automatique de valeurs	**IDENTITY NOT NULL PRIMARY KEY** La génération de valeurs débute à 1 et suivent des valeurs incrémentées de 1
3.	{FK}	Cet attribut constitue une clé étrangère avec intégrité référentielle en ajout	**REFERENCES** *Table_mère***(***Clé_mère***)** Le type de données de la clé primaire doit être le même que celui de la clé étrangère
4.	Multiplicité *minimale* 1 côté **mère**	La clé étrangère comporte obligatoirement une valeur	**NOT NULL** pour la clé étrangère qui réalise l'association
6.	Multiplicité *maximale* 1 côté **mère**	Une ligne fille ne peut être associée au plus qu'à une ligne mère	Aucune clause requise car dans le modèle relationnel un attribut, donc une clé étrangère, ne peut avoir qu'une seule valeur à la fois
12.	{>*Nom_attribut*$_2$}	Cet attribut possède une contrainte voulant que sa valeur soit limitée par la valeur d'un attribut sur la même ligne	**CONSTRAINT** *Nom_contrainte* **CHECK(***Nom_attribut* > *Nom_attribut*$_2$**)**
13.	{<= *Valeur hors ligne*}	Cet attribut possède une contrainte voulant que sa valeur soit limitée par la valeur d'un attribut sur une autre ligne de la même table ou une ligne d'une autre table	**CONSTRAINT** *Nom_contrainte* **CHECK(NOT EXISTS (SELECT** *Nom_attribut* **FROM** *Jointure* **WHERE** *Condition***))**

quant à la création de la table **Employé**. Les exigences de type 3, 4 et 6 ont déjà été évoquées dans les exemples précédents. Le modèle 3-3 permet d'illustrer la mise en œuvre de trois nouveaux types d'exigences qui sont numérotés 11, 12 et 13.

Table avec clé primaire composée

Lorsque qu'un modèle relationnel n'a pas été complètement optimisé, il peut comporter des tables dont la clé primaire est composée. Le script pour réaliser une clé primaire composée ne doit pas être de même nature que celui pour réaliser une clé primaire simple. Il devra comporter obligatoirement une clause **PRIMARY KEY** ayant comme paramètres les attributs de la clé.

FIGURE 3-4 **Modèle relationnel comportant une table avec clé primaire composée**



Le modèle de la figure 3-4 nous permettra d'illustrer comment le script du modèle physique doit réaliser une clé primaire composée mais aussi comment l'intégrité référentielle (en ajout, suppression et mise à jour) peut être mise en œuvre sur chaque clé étrangère (mention {Cascade}).

Les tables mères sont créées les premières et les exigences du modèle sont formulées dans le script selon les conventions décrites dans l'exemple 3-1.

```
CREATE TABLE Magasin
([No agence] SMALLINT NOT NULL PRIMARY KEY,
[No civique] VARCHAR(10),
[Rue] VARCHAR(30),
[Ville] VARCHAR(30),
[Code postal] VARCHAR(6) NOT NULL,
[No téléphone] VARCHAR(10));
```

```
CREATE TABLE Employé
([No employé] SMALLINT NOT NULL PRIMARY KEY,
[Nom employé] VARCHAR(20),
[Prénom employé] VARCHAR(20),
[Salaire employé] CURRENCY NOT NULL,
CONSTRAINT SalaireSupÀZéro CHECK([Salaire employé]>0))
```

La table fille **Embauche** est ensuite créée. On peut noter dans le script qui suit, la réalisation des deux clés étrangères avec les clauses **ON DELETE CASCADE** et **ON UPDATE CASCADE** pour spécifier qu'il s'agit dans chaque cas d'assurer l'intégrité référentielle en ajout, en suppression et en mise à jour. La suppression d'une ligne mère entraîne la suppression de toutes ses lignes filles. La mise à jour de la clé primaire d'une ligne mère entraîne la mise à jour des valeurs des clés étrangères correspondantes dans les lignes filles.

```
CREATE TABLE Embauche
([No agence] SMALLINT NOT NULL
REFERENCES Magasin([No agence]) ON DELETE CASCADE
ON UPDATE CASCADE,
[No employé] SMALLINT NOT NULL
REFERENCES Employé([No employé]) ON DELETE CASCADE
ON UPDATE CASCADE,
[Date embauche] DATE,

PRIMARY KEY([No agence],[No employé]))
```

La clé primaire composée doit être réalisée grâce à une clause **PRIMARY KEY** faisant état des deux attributs composant la clé, attributs donnés en paramètres.

```
PRIMARY KEY([No agence],[No employé])
```

Au tableau des exigences, on se doit d'ajouter deux nouveaux types d'exigences, soit le type 14 pour la formulation des clés primaires composées et le type 15 pour la codification de l'intégrité référentielle en ajout, suppression et mise à jour sur une clé étrangère. Voir le tableau 3-6.

TABLEAU 3-6 **Résumé des clauses du script de création de tables requises pour réaliser le modèle physique correspondant au modèle 3-4**

Exigence	Niveau logique	Signification	Niveau physique : Clause requise dans le script
14.	*Nom_attribut*$_1$ {PK} … *Nom_attribut*$_n$ {PK}	Ces attributs forment une clé primaire composée : aucun de ces attributs ne peut avoir une valeur nulle et leur combinaison ne peut avoir de doublons (intégrité d'entité)	**PRIMARY KEY (***Nom_attribut*$_1$*, …, Nom_attribut*$_n$**)** Chaque attribut a été nommé plus tôt dans le script et porte une clause **NOT NULL**
2.	{Non nul}	La valeur de cet attribut ne peut être nulle (laissée en blanc)	**NOT NULL**
15.	{FK}{Cascade}	Cet attribut constitue une clé étrangère avec intégrité référentielle en ajout, suppression et mise à jour	**REFERENCES** *Table_mère***(***Clé_mère***) ON DELETE CASCADE ON UPDATE CASCADE** Le type de données de la clé primaire doit être le même que celui de la clé étrangère
4.	Multiplicité *minimale* 1 côté **mère**	La clé étrangère comporte obligatoirement une valeur	**NOT NULL** pour la clé étrangère qui réalise l'association
5.	Multiplicité *minimale* 1 côté **fille**	On doit avoir au moins une ligne de la table fille associée à une ligne de la table mère	Ne pas mettre en œuvre avec **CREATE TABLE** car il y a un conflit avec l'intégrité référentielle
6.	Multiplicité *maximale* 1 côté **mère**	Une ligne fille ne peut être associée au plus qu'à une ligne mère	Aucune clause requise car dans le modèle relationnel une clé étrangère ne peut avoir qu'une seule valeur à la fois
7.	{>*Nombre*}	Cet attribut possède une contrainte de domaine	**CONSTRAINT** *Nom_contrainte* **CHECK(***Nom_attribut***>***Nombre***)**

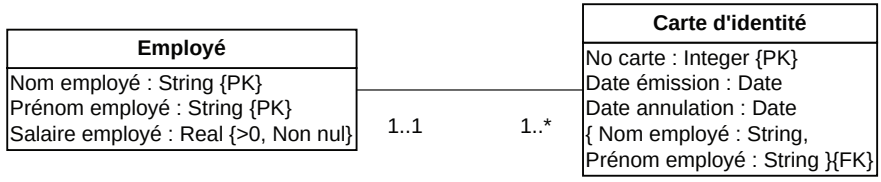
Table avec clé étrangère composée

Bien que cette situation soit exceptionnelle, il peut arriver que la clé étrangère qui associe une table fille à une table mère soit composée de deux attributs ou plus. Il s'agit d'une conséquence du choix d'un identifiant composé pour une entité forte. Nous avons vu au chapitre 1 que ce type d'identifiant n'est pas souhaitable mais il n'est pas interdit d'y faire appel. Un tel identifiant entraîne une clé primaire composée pour la table dérivée.

S'il s'avère que la table dérivée est une table mère, la table fille associée doit recevoir une copie de la clé primaire composée à titre de clé étrangère composée. Dans le cas où aucune mesure d'optimisation ne serait prise avant

la création du modèle physique, le script doit implanter une clé étrangère composée par le biais de la clause **FOREIGN KEY**. La figure 3-5 illustre une telle situation. La clé primaire de la table **Employé** est composée de deux attributs. Elle est associée à la table **Carte d'identité** pour laquelle elle agit comme table mère. La clé primaire composée est reproduite dans la table **Carte d'identité** et ses deux attributs sont placés entre accolades, **{Nom employé:** `String`**, Prénom employé:**`String`**}**, suivis de la mention {FK} pour bien mettre en évidence qu'il s'agit là d'une clé étrangère composée et non pas de deux clés étrangères.

FIGURE 3-5 **Modèle relationnel comportant une table avec clé étrangère composée**



La table **Employé** doit être créée avec le script qui suit. On y retrouve une clause **PRIMARY KEY** qui codifie la clé primaire composée. Les deux attributs qui composent la clé ont été déclarés plus tôt et portent chacun une clause **NOT NULL**, assurant ainsi la contrainte d'intégrité d'entité.

```
CREATE TABLE Employé
([Nom employé] VARCHAR(20) NOT NULL,
[Prénom employé] VARCHAR(20) NOT NULL,
[Salaire employé] CURRENCY NOT NULL,
CONSTRAINT SalaireSupÀZéro CHECK([Salaire employé]>0),

PRIMARY KEY([Nom employé],[Prénom employé]))
```

La table **Carte d'entité** est créée ensuite. La clause **FOREIGN KEY** formule les exigences d'une clé étrangère composée. On a pris soin auparavant de déclarer chaque attribut composant la clé étrangère et portant la clause **NOT NULL** pour implanter la multiplicité minimale 1 du côté de la table mère **Employé**.

```
CREATE TABLE [Carte d'identité]
([No carte] SMALLINT NOT NULL PRIMARY KEY,
[Date émission] DATE,
[Date annulation] DATE,
[Nom employé] VARCHAR(20) NOT NULL,
[Prénom employé] VARCHAR(20) NOT NULL,

FOREIGN KEY([Nom employé],[Prénom employé])
REFERENCES Employé([Nom employé],[Prénom employé]) )
```

Le tableau 3-7 compile les exigences de création des tables du modèle 3-5 où apparaît le type 16 concernant la création d'une clé étrangère composée.

Les exemples 3-1 à 3-5 nous ont permis de constater toute la souplesse de l'instruction SQL **CREATE TABLE** pour la création d'un schéma physique, tout en démontrant sa complexité. La prochaine section présente une description formelle de l'instruction **CREATE TABLE** et des clauses qui la composent.

Syntaxe formelle de l'instruction CREATE TABLE

La syntaxe formelle d'une instruction appartenant à un langage de programmation est habituellement définie sous une forme appelée la notation *Backus Naur Form (BNF)*. Elle permet d'exprimer succinctement la syntaxe générale d'une instruction et de valider si une instance de l'instruction s'y conforme.

Toute instruction SQL est constituée de *mots-clés* et de *mots définis par l'utilisateur*. Les mots-clés sont écrits en caractères majuscules et souvent en gras. **CHECK** est un mot-clé comme tous les noms des clauses présentes dans une instruction **CREATE TABLE**. Les mots définis par l'utilisateur sont choisis par l'utilisateur et réfèrent à des objets de la base de données : une table, une clé primaire simple, une colonne, etc.

La notation BNF prescrit les règles suivantes pour la définition formelle de la syntaxe d'une instruction :

- Les lettres majuscules représentent des mots-clés qui doivent s'écrire de la manière strictement indiquée ;
- Les lettres minuscules représentent les mots définis par l'utilisateur ;

TABLEAU 3-7 **Résumé des clauses du script de création de tables requises pour réaliser le modèle physique correspondant au modèle 3-5**

Exigence	Niveau logique	Signification	Niveau physique : Clause requise dans le script
14.	*Nom_attribut*$_1$ {PK} … *Nom_attribut*$_n$ {PK}	Ces attributs forment une clé primaire composée : aucun de ces attributs ne peut avoir une valeur nulle et leur combinaison ne peut avoir de doublons (intégrité d'entité)	**PRIMARY KEY (***Nom_attribut*$_1$*, …, Nom_attribut*$_n$**)** Chaque attribut a été nommé plus tôt dans le script et porte une clause **NOT NULL**
2.	{Non nul}	La valeur de cet attribut ne peut être nulle (laissée en blanc)	**NOT NULL**
16.	{*Nom_attribut*$_1$ *: Type, …, Nom_attribut*$_n$ *: Type*}{FK}	Ce groupe d'attributs constituent une clé étrangère avec intégrité référentielle en ajout	**FOREIGN KEY (***Nom_attribut*$_1$*, …, Nom_attribut*$_n$**) REFERENCES** *Table_mère(Clé_attribut*$_1$*, …, Clé_attribut*$_n$*)* Le type de données de la clé primaire doit être le même que celui de la clé étrangère
4.	Multiplicité *minimale* 1 côté **mère**	La clé étrangère comporte obligatoirement une valeur	**NOT NULL** pour la clé étrangère qui réalise l'association
5.	Multiplicité *minimale* 1 côté **fille**	On doit avoir au moins une ligne de la table fille associée à une ligne de la table mère	Ne pas mettre en œuvre avec **CREATE TABLE** car il y a un conflit avec l'intégrité référentielle
6.	Multiplicité *maximale* 1 côté **mère**	Une ligne fille ne peut être associée au plus qu'à une ligne mère	Aucune clause requise car dans le modèle relationnel une clé étrangère ne peut avoir qu'une seule valeur à la fois
7.	{>*Nombre*}	Cet attribut possède une contrainte de domaine	**CONSTRAINT** *Nom_contrainte* **CHECK(***Nom_attribut***>***Nombre***)**

- Les mots précédés de < et se terminant avec >, généralement en italique, représentent une sous-clause de l'instruction qui comporte sa propre syntaxe ;
- La barre verticale (|) indique un choix parmi une alternative ;
- Les accolades indiquent la présence obligatoire d'un élément ;
- Les crochets indiquent la présence facultative d'un élément ;
- Les points de suspension (…) indiquent une répétition optionnelle d'un élément. Le cas échéant, chaque élément est séparé du précédent par une virgule.

Par convention, le libellé des mots définis par l'utilisateur fait appel à la terminologie des BD relationnelles : table (au lieu de relation), colonne (au lieu d'attribut), ligne (au lieu de tuple), clé primaire, clé étrangère et ainsi de suite.

La syntaxe formelle de l'instruction **CREATE TABLE** est donnée ci-après. Elle répond à une version simplifiée de la syntaxe suggérée par les normes ANSI-92 et ISO de manière à ce que la grande majorité des SGDB relationnels puissent l'interpréter adéquatement.

```
CREATE TABLE NomDeTable
 ({NomDeColonne <TypeDeDonnée>
 [DEFAULT {Littéral | NULL}]
 [NOT NULL]
 [UNIQUE]
 [PRIMARY KEY]
 [REFERENCES TableMère [(ColTableMère)]
 [<ActionRéférentielle>]]} [,…]
 [, PRIMARY KEY(NomDeColonne [,…])]
 [, UNIQUE (NomDeColonne [,…])][…]
 [, FOREIGN KEY(NomDeColonne [,…])
 REFERENCES TableMère(ColTableMère [,…])
 [<ActionRéférentielle>]] […]
 [, CONSTRAINT NomContrainte CHECK(<Condition>)][…] )
```

Cette instruction crée une table portant le nom `NomDeTable` dotée de colonnes dont le type de données doit obligatoirement être spécifié pour chacune. La clause optionnelle **DEFAULT** permet de définir la valeur par défaut d'une colonne. Ce ne peut être qu'un *littéral*, soit un nombre ou une chaîne de caractères. Les clauses **NOT NULL** et **UNIQUE** sont requises si on souhaite, d'une part, que la colonne ne puisse être laissée en blanc et qu'elle ne puisse recevoir de doublons, d'autre part.

Clé primaire simple

La clause **PRIMARY KEY**, inscrite à la suite du type de données de la colonne, fait de cette colonne la *clé primaire simple* de la table. Il importe de souligner que cette clause signifie que la colonne n'aura pas de doublons et aura une valeur non nulle par défaut. Les clauses **UNIQUE** et **NOT NULL** seraient donc redondantes dans le cas d'une clé primaire simple. Puisqu'une seule clé primaire est présente dans une table, la clause **PRIMARY KEY** ne peut être utilisée

qu'une seule fois dans l'instruction **CREATE TABLE**. Néanmoins une clé secondaire peut exister en parallèle. La *clé secondaire simple* doit être dotée des clauses **UNIQUE** et **NOT NULL.**

Clé étrangère simple

La clause **REFERENCES**, inscrite à la suite du type de données de la colonne, fait de cette colonne une *clé étrangère simple* dans la table. Elle indique la table mère, `TableMère`, et entre parenthèses la colonne de la table mère, `ColTableMère`, qui assure l'association avec sa table fille `NomDeTable` grâce à la colonne `NomDeColonne`. Dans le cas où `ColTableMère` n'est pas précisé, il doit exister une colonne dans la table mère qui porte le nom `NomDeColonne`. Il est en outre essentiel que `ColTableMère` et `NomDeColonne` aient le même type de données et que la table `NomDeTable` soit créée au préalable. Cette clause peut être suivie de clauses précisant des actions référentielles, c'est-à-dire quelles actions peuvent être entreprises à la suite de la suppression d'une ligne de la table mère qui est associée à des lignes de la table fille, ou encore à la suite de la mise à jour de la valeur de la clé primaire. La sous-clause *<ActionRéférentielle>* se définit comme suit :

```
<ActionRéférentielle> =
{ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL} |
 ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}}
```

ON DELETE spécifie l'action à mener quand une ligne est supprimée dans la table mère. Selon la clause qui la suit immédiatement, l'action enclenchée automatiquement est variable :

- **NO ACTION**, rien n'est fait ;
- **CASCADE**, les lignes associées dans les tables filles sont supprimées et en cascade toutes les lignes associées dans leurs propres tables filles ;
- **SET DEFAULT**, la valeur de la clé étrangère de chaque ligne fille est ramenée à la valeur par défaut de la colonne ;
- **SET NULL**, la valeur de la clé étrangère de chaque ligne fille est mise à **NULL**.

ON UPDATE spécifie l'action à mener quand une ligne de la table mère voit la valeur de sa clé primaire modifiée. Selon la clause qui la suit immédiatement, l'action enclenchée automatiquement est variable :

- **NO ACTION**, rien n'est fait ;
- **CASCADE**, les lignes associées dans les tables filles voient la valeur de la clé étrangère modifiée et en cascade toute ligne associée dans leurs propres tables filles voit la valeur de la clé étrangère modifiée ;

- **SET DEFAULT**, la valeur de la clé étrangère de chaque ligne fille est ramenée à la valeur par défaut de la colonne ;
- **SET NULL**, la valeur de la clé étrangère de chaque ligne fille est mise à **NULL**.

Clé secondaire composée

Les clauses **UNIQUE** et **PRIMARY KEY** peuvent être aussi appliquées à un groupe de colonnes. **UNIQUE (**`NomDeColonne [,…]`**)** sera utile pour spécifier une *clé secondaire composée*. **PRIMARY KEY(**`NomDeColonne [,…]`**)** le sera pour une *clé primaire composée*, mais encore faut-il qu'il n'y ait pas déjà une clause **PRIMARY KEY** dans la même instruction **CREATE TABLE**.

Clé étrangère composée

La clause **FOREIGN KEY** peut être utilisée pour définir une *clé étrangère composée*. La liste des colonnes entre parenthèses forme solidairement une clé étrangère et on doit retrouver du côté de la table mère le même nombre de colonnes. Chaque colonne dans la liste des colonnes `ColTableMère` de la table mère donnée dans la sous-clause **REFERENCES** doit disposer du même type de données que la colonne correspondante dans la liste des `NomDeColonne` qui suit la clause **FOREIGN KEY**. Des actions référentielles, de la même nature que celles vues plus tôt pour une clé étrangère simple, peuvent être appliquées solidairement sur les colonnes de la clé étrangère composée.

Contraintes générales

Une contrainte générale appartient à un des cinq types suivants :

1. une contrainte appliquée à une seule colonne de la table, par exemple une contrainte de domaine qui restreint les valeurs admissibles dans cette colonne ;
2. une contrainte mettant en cause plus d'une colonne de la même table ;
3. une contrainte mettant en cause des colonnes de plusieurs tables différentes liées par une opération de jointure ;
4. une contrainte stipulant l'existence de lignes provenant de la même table ou de tables différentes liées par une opération de jointure ;
5. une contrainte stipulant l'existence d'un nombre précis de lignes provenant de la même table ou de tables différentes liées par une opération de jointure.

La contrainte porte obligatoirement un nom et une condition dont la formulation peut s'avérer complexe et ardue. Nous étudions, sous la rubrique qui suit, la formulation de contraintes générales du type 1 ou 2 uniquement. Les contraintes du type 3, 4 ou 5 font appel à des conditions mettant en œuvre une instruction SQL **SELECT** qui peut être fort élaborée, comme dans l'exemple étudié plus tôt.

```
CONSTRAINT SalaireInfEgalAuPatron CHECK(NOT EXISTS
(SELECT Employé.[Salaire employé]
FROM Employé INNER JOIN Employé AS Employé_1 ON
Employé.[No employé]=Employé_1.[No employé patron]
WHERE (((Employé.[Salaire employé])<Employé_1.[Salaire
employé])))
```

La rédaction des contraintes de type 3, 4 ou 5 exige une excellente maîtrise des requêtes de sélection en mode SQL et nous reportons le lecteur au chapitre 5 de [ConB 05] pour une couverture complète de ce sujet.

Contraintes générales de type 1 ou 2

Les contraintes générales de type 1 ou 2 font appel à une sous-clause <*Condition*> dont la syntaxe générale est définie de la façon récursive suivante :

```
<Condition> =
{<Valeur> <Opérateur> { <Valeur> | (<Sélection_Un>)}
| <Valeur> [NOT] BETWEEN <Valeur> AND <Valeur>
| <Valeur> [NOT] LIKE <Valeur>
| <Valeur> [NOT] IN (<Valeur> [, …] | <Sélection_Liste>)
| <Valeur> IS [NOT] NULL
| <Valeur> { {= | < | >} | >= | <=}
{ALL | SOME | ANY} (<Sélection_Liste>)
| <Valeur> [NOT] CONTAINNING <Valeur>
| <Valeur> [NOT] STARTING <Valeur>
| ( <Condition> )
| NOT <Condition>
| <Condition> OR <Condition>
| <Condition> AND <Condition> }
```

```
<Valeur> =
NomDeColonne | <Constante> | <ExpressionArithmétique>
```

```
<Constante> =
Nombre | 'Chaîne de caractères'
```

```
<Opérateur> =
{= | < | > | <= | >= | }
```

```
<ExpressionArithmétique> =
Toute expression arithmétique valide comportant
uniquement des constantes et des Noms De Colonnes
```

```
<Sélection_Un> =
Une instruction SELECT sur une colonne retournant qu'une
seule valeur
```

```
<Sélection_Liste> =
Une instruction SELECT sur une colonne retournant zéro ou
plusieurs valeurs
```

Voici quelques exemples de contraintes correctement formulées sur le plan syntaxique. Sur le plan opérationnel, la contrainte ne produira le résultat souhaité que si les noms des colonnes dans la clause sont tous des noms présents dans la table créée à l'aide d'une instruction **CREATE TABLE** qui incorpore la clause **CONSTRAINT**.

```
CONSTRAINT FinApresDébut CHECK([Date fin]>[Date début])
```

```
CONSTRAINT FinApresDébut2 CHECK([Date fin] IS NOT NULL
AND [Date début] IS NOT NULL AND [Date fin]>[Date début])
```

Dans l'exemple suivant, 0,10 est une constante numérique valide.

```
CONSTRAINT CommissionÀDix CHECK([Commission]=[Montant de
vente]*0,10)
```

Ci-après, *[Date dernière augmentation]+365* est une expression arithmétique valide, tout comme *[Montant de vente]*0,10* dans l'exemple précédent.

```
CONSTRAINT AugmentationDansUnAn CHECK([Date prochaine
augmentation] = [Date dernière augmentation]+365)
```

```
CONSTRAINT LimitesSalaire CHECK([Salaire] BETWEEN 10000
AND 100000)
```

```
CONSTRAINT LimitesSalaire2 CHECK([Salaire] >= 10000 AND
[Salaire] <= 100000)
```

Les chaînes de caractères 'Visa', 'MasterCard', 'American Express' et 'Discovery' sont des constantes sur le plan syntaxique.

```
CONSTRAINT TypeCarteValide CHECK([Type carte crédit] IN
('Visa', 'MasterCard', 'American Express', 'Discovery'))
```

Dans les deux exemples qui suivent, le nom de la colonne qui suit le mot-clé **SELECT** doit obligatoirement appartenir à la table dont le nom est donné après le mot-clé **FROM**.

L'instruction **SELECT** incorporée à la clause **CONSTRAINT** du prochain exemple retourne comme résultat une liste de tous les noms de villes présents dans la table **Tables des villes**.

```
CONSTRAINT VilleValide CHECK([Ville] IN (SELECT [Nom
ville] FROM [Table des villes]))
```

L'instruction **SELECT** incorporée à la clause **CONSTRAINT** produit comme résultat une seule valeur numérique dans ce prochain exemple.

```
CONSTRAINT LimiteValide CHECK([Limite du client] <=
(SELECT [Limite maximale crédit] FROM [Table des montants
limites]))
```

Il existe une instruction SQL qui est la contrepartie de **CREATE TABLE**. Son nom est **DROP TABLE** et elle permet de supprimer complètement une table. Cette opération devient nécessaire si la table n'a plus sa raison d'être dans la BD ou si sa structure doit être complètement remise en question. La syntaxe formelle de **DROP TABLE** est la suivante :

```
DROP TABLE NomDeTable
```

Cette instruction supprime la table nommée, et bien sûr son contenu, ce qui peut mener à des pertes de données à moins d'avoir exporté ces données au préalable dans une autre table de la même BD ou située hors de la BD.

Pour assurer la cohérence de la BD, une table ne peut être effacée si elle agit comme table mère pour d'autres tables. Les tables filles doivent d'abord être modifiées pour éliminer les clés étrangères qui réfèrent à la table mère avant que cette dernière ne puisse être effacée. La suppression d'une clé étrangère présente dans une table peut être réalisée avec l'instruction **ALTER TABLE**. Celle-ci permet d'apporter par ailleurs de nombreux types de modifications à une table.

Syntaxe formelle de l'instruction ALTER TABLE

La structure d'une table peut évoluer dans le temps. Il est fréquent qu'une clé étrangère s'ajoute à la table ou doive être supprimée comme nous l'évoquions plus tôt. Il peut être nécessaire d'ajouter une nouvelle colonne ou de changer les propriétés d'une colonne existante : type de données, valeur par défaut, et ainsi de suite. Aussi les normes ANSI-92 et ISO prévoient-elles une instruction SQL appelée **ALTER TABLE**. À l'aide de cette instruction, on peut réaliser les opérations suivantes sur une table :

- ajout d'une colonne
- suppression d'une colonne
- suppression de la valeur par défaut pour une colonne
- changement de la valeur par défaut pour une colonne
- ajout d'une contrainte
- suppression de contrainte

La syntaxe formelle de **ALTER TABLE** est la suivante :

```
ALTER TABLE NomDeTable
 {ADD COLUMN NomDeColonne <TypeDeDonnée>
 [DEFAULT {Littéral | NULL}]
 [NOT NULL]
 [UNIQUE]
 [PRIMARY KEY]
 [REFERENCES TableMère [(ColTableMère)]
 [<ActionRéférentielle>]]
 | DROP NomDeColonne
 | ALTER COLUMN NomDeColonne DROP DEFAULT
 | ALTER COLUMN NomDeColonne SET DEFAULT {Littéral | NULL}
 | ADD CONSTRAINT NomContrainte CHECK(<Condition>)
 | DROP CONSTRAINT }
```

Certains SGBD dont MS Access ne permettent pas, avec l'instruction **ALTER TABLE DROP**, la suppression d'une colonne qui fait l'objet d'un index. C'est le cas des clés primaires simples et composées et des colonnes créées avec la clause **UNIQUE**. Il faut au préalable supprimer l'index à l'aide de l'instruction **DROP INDEX**. Nous reviendrons sous peu sur cette instruction.

Les exemples suivants illustrent les diverses possibilités de l'instruction **ALTER TABLE**.

Ajout d'une clé étrangère simple :

```
ALTER TABLE Commande
ADD COLUMN [No client] INTEGER NOT NULL
REFERENCES Client([No client]) ON DELETE CASCADE
```

Ajout d'une clé primaire simple :

```
ALTER TABLE Commande
ADD COLUMN [No commande] INTEGER NOT NULL DEFAULT 0
PRIMARY KEY
```

Suppression d'une colonne :

```
ALTER TABLE Commande
DROP COLUMN [Date commande]
```

Suppression de la valeur par défaut :

```
ALTER TABLE Commande
ALTER COLUMN [No commande] DROP DEFAULT
```

Changement de la valeur par défaut :

```
ALTER TABLE Commande
ALTER COLUMN [No commande] SET DEFAULT 0
```

Ajout d'une contrainte à une table :

```
ALTER TABLE Commande
ADD CONSTRAINT TypeCarteValide CHECK([Type carte crédit]
IN ('Visa', 'MasterCard', 'American Express',
'Discovery'))
```

Suppression d'une contrainte dans une table :

```
ALTER TABLE Commande
DROP CONSTRAINT TypeCarteValide
```

Certains SGBD dont MS Access ne permettent pas la suppression d'une table avec l'instruction **DROP TABLE** à moins que toutes les contraintes qui portent sur cette table n'aient été supprimées avec **ALTER TABLE** `NomDeTable` **DROP CONSTRAINT**.

Syntaxe formelle de l'instruction CREATE INDEX

Un *index* est une composante physique de la BD qui permet d'accélérer l'accès aux données. Il joue le même rôle que l'index d'un livre où un mot est directement relié à la page ou aux pages où il apparaît. Un index dans une BD permet d'identifier aussitôt sur une ou des lignes si une valeur donnée est présente dans une colonne.

La présence d'un index peut accélérer grandement la performance des requêtes, notamment les requêtes de sélection qui font appel à plusieurs tables. Il y a cependant un prix à payer pour cela : le SGBD doit assurer l'entretien de chaque index, ce qui signifie que l'index est modifié à chaque fois qu'une valeur est changée dans une colonne appartenant à l'index. Dans le cas des colonnes *indexées sans doublons* créées avec la clause **UNIQUE**, le SGBD doit en plus assurer l'unicité des valeurs de la colonne ou de la combinaison de colonnes en rejetant tout doublon. La gestion de l'index se fera plus lourde au fur et a mesure que la taille de la table augmente.

La performance de la gestion de l'index dépend aussi du nombre de colonnes composant l'index. Comme chaque clé primaire et chaque clé secondaire exige son propre index, le nombre d'index dans une BD peut être relativement élevé et leur gestion coûteuse sur le plan de l'efficacité s'il s'agit de clés composées. C'est la raison pour laquelle nous proposions au chapitre précédent d'optimiser les clés primaires des tables en remplaçant toute clé primaire composée par une clé primaire simple à génération automatique de valeurs.

Le concepteur doit être conscient qu'un index est créé pour répondre à toute situation. La gestion d'un index est d'autant plus lourde qu'il comporte plusieurs colonnes et exclut les doublons :

- instruction **CREATE TABLE** utilisant la clause **PRIMARY KEY** et/ou **UNIQUE** pour une colonne
- instruction **CREATE TABLE** utilisant la clause **PRIMARY KEY** portant sur un groupe de colonnes
- instruction **CREATE TABLE** utilisant la clause **UNIQUE** portant sur un groupe de colonnes
- instruction **CREATE INDEX** portant sur une colonne ou un groupe de colonnes

L'instruction **CREATE INDEX** possède une syntaxe simple :

```
CREATE INDEX [UNIQUE] NomDeIndex
 ON NomDeTable (NomDeCollonne [, …] )
```

La clause **UNIQUE** en fait un index sans doublons.

La suppression d'un index exige l'instruction **DROP INDEX** :

```
DROP INDEX NomDeIndex
```

Les index qui sont créés automatiquement par une instruction **CREATE TABLE** ne portent pas un nom explicite. Aussi est-il impossible de les supprimer avec l'instruction **DROP INDEX**. Cela pose un problème au concepteur d'une BD lorsque le SGBD refuse de supprimer une colonne avec **DROP COLUMN** parce qu'elle fait partie d'un index. Il se trouve alors dans un cul de sac. La colonne ne peut être supprimée avant la suppression de l'index mais ce dernier ne peut être supprimé avec **DROP INDEX** car son nom est inconnu.

Heureusement, la plupart les SGBD possèdent une interface graphique qui permet de supprimer une colonne dans une table ainsi que l'index auquel elle appartient, en sélectionnant le nom de la colonne dans une fenêtre et en appelant au menu `Edition->Supprimer`.

RÉALISATION DU MODÈLE PHYSIQUE EN SQL

L'introduction au langage SQL pour la définition des données nous donne les éléments nécessaires à la réalisation du modèle physique par le biais d'un script d'instructions **CREATE TABLE**. Nous avons mis l'accent sur deux éléments fondamentaux :

1. Les exigences qui sont présentes dans le modèle relationnel et les clauses de l'instruction **CREATE TABLE** qui permettent de les réaliser, telles que résumées dans le tableau 3-8;
2. La syntaxe formelle de l'instruction **CREATE TABLE** de même que celle des instructions accessoires que sont **DROP TABLE**, **ALTER TABLE**, **CREATE INDEX** et **DROP INDEX**.

Dans cette section, nous proposons une démarche de traduction du modèle relationnel en un modèle physique basée sur la satisfaction des exigences formulées au niveau logique et le respect strict de la syntaxe de **CREATE TABLE**.

La démarche vise à s'assurer que la séquence de création des tables tient compte de la logique de l'instruction **CREATE TABLE**, notamment qu'une table mère doit être créée avant ses tables filles.

Elle se résume ainsi:

1. Créer les tables mères qui ne soient pas à la fois table fille (Aucune clé étrangère). Ignorer les associations.
2. Créer les tables mères qui agissent aussi comme table fille pour certaines associations. Pour chaque association où la table mère agit comme table fille, prévoir les clauses requises pour assurer les exigences 4 et 9. Débuter avec les tables comportant le plus d'associations où elles agissent comme tables mères et le moins d'associations où elles agissent comme tables filles.
3. Créer les autres tables filles. Pour chaque association prévoir les clauses requises pour assurer les exigences 4 et 9.
4. Dénombrer le nombre de tables du modèle relationnel. Vérifier que toutes les tables ont été créées.

Pour illustrer la démarche, nous vous proposons une série d'études de cas qui s'inscrivent dans le cadre de référence présenté dans le chapitre précédent.

TABLEAU 3-8 **Exigences formulées au niveau logique et clauses du script de création de tables requises pour les satisfaire**

Exigence	Niveau logique	Signification	Niveau physique : Clause requise dans le script
1.	{PK}	Cet attribut constitue la clé primaire : il ne peut avoir une valeur nulle et avoir des doublons (intégrité d'entité)	**NOT NULL PRIMARY KEY**
2.	{Non nul}	La valeur de cet attribut ne peut être nulle (laissée en blanc)	**NOT NULL**
3.	{FK}	Cet attribut constitue une clé étrangère avec intégrité référentielle en ajout	**REFERENCES** *Table_mère*(*Clé_mère*) Le type de données de la clé primaire doit être le même que celui de la clé étrangère
4.	Multiplicité *minimale* 1 côté **mère**	La clé étrangère comporte obligatoirement une valeur	**NOT NULL** pour la clé étrangère qui réalise l'association
5.	Multiplicité *minimale* 1 côté **fille**	On doit avoir au moins une ligne de la table fille associée à une ligne de la table mère	Ne pas mettre en œuvre avec **CREATE TABLE** car il y a un conflit avec l'intégrité référentielle
6.	Multiplicité *maximale* 1 côté **mère**	Une ligne fille ne peut être associée au plus qu'à une ligne mère	Aucune clause requise car dans le modèle relationnel un attribut, donc une clé étrangère, ne peut avoir qu'une seule valeur à la fois
7.	{>*Nombre*}	Cet attribut possède une contrainte de domaine	**CONSTRAINT** *Nom_contrainte* **CHECK(***Nom_attribut***>***Nombre***)**
8.	{Défaut *valeur*}	Cet attribut dispose d'une valeur par défaut	**DEFAULT** *Valeur_par_défaut*
9.	Multiplicité *maximale* 1 côté **fille**	La clé étrangère ne peut avoir des doublons	**UNIQUE** pour la clé étrangère qui réalise l'association
10.	enum{val_1, val_2, …, val_n}	Cet attribut possède une contrainte de domaine	**CONSTRAINT** *Nom_contrainte* **CHECK(***Nom_attribut* **IN (**val_1, val_2, …, val_n**))**
11.	{PK auto}	Cet attribut constitue une clé primaire avec génération automatique de valeurs	**IDENTITY NOT NULL PRIMARY KEY** La génération de valeurs débute à 1 et suivent des valeurs incrémentées de 1
12.	{>$Nom\_attribut_2$}	Cet attribut possède une contrainte voulant que sa valeur soit limitée par la valeur d'un attribut sur la même ligne	**CONSTRAINT** *Nom_contrainte* **CHECK(***Nom_attribut* > $Nom\_attribut_2$**)**

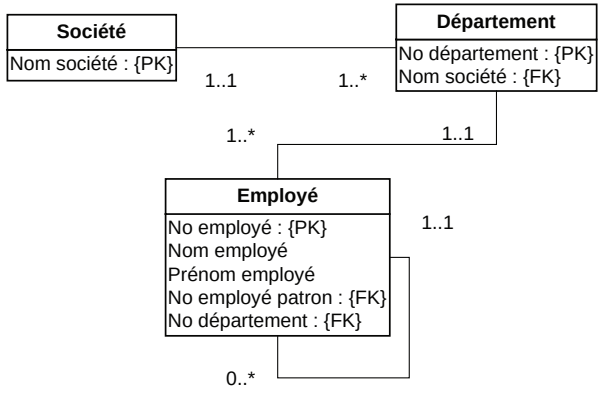
TABLEAU 3-8 **Exigences formulées au niveau logique et clauses du script de création de tables requises pour les satisfaire (*suite*)**

Exigence	Niveau logique	Signification	Niveau physique : Clause requise dans le script
13.	{<= *Valeur hors ligne*}	Cet attribut possède une contrainte voulant que sa valeur soit limitée par la valeur d'un attribut sur une autre ligne de la même table ou une ligne d'une autre table	**CONSTRAINT** *Nom_contrainte* **CHECK(NOT EXISTS (SELECT** *Nom_attribut* **FROM** *Jointure* **WHERE** *Condition*))
14.	*Nom_attribut*$_1$ {PK} … *Nom_attribut*$_n$ {PK}	Ces attributs forment une clé primaire composée : aucun de ces attributs ne peut avoir une valeur nulle et leur combinaison ne peut avoir de doublons (intégrité d'entité)	**PRIMARY KEY(***Nom_attribut*$_1$, …, *Nom_attribut*$_n$**)** Chaque attribut a été nommé plus tôt dans le script et porte une clause **NOT NULL**
15.	{FK}{Cascade}	Cet attribut constitue une clé étrangère avec intégrité référentielle en ajout, suppression et mise à jour	**REFERENCES** *Table_mère***(***Clé_mère***)** ON DELETE CASCADE ON UPDATE CASCADE Le type de données de la clé primaire doit être le même que celui de la clé étrangère
16.	{*Nom_attribut*$_1$: *Type*, …, *Nom_attribut*$_n$: *Type*}{FK}	Ce groupe d'attributs constituent une clé étrangère avec intégrité référentielle en ajout	**FOREIGN KEY (***Nom_attribut*$_1$, …, *Nom_attribut*$_n$**) REFERENCES** *Table_mère***(***Clé_attribut*$_1$, …, *Clé_attribut*$_n$**)** Le type de données de la clé primaire doit être le même que celui de la clé étrangère
17.	{Unique}	Attribut sans doublons	**UNIQUE**
18.	{PK, FK} L'attribut est le seul attribut de la clé primaire ; la syntaxe de **CREATE TABLE** ne permet pas simultanément de définir une clé primaire simple qui est aussi clé étrangère	Cet attribut est une clé primaire simple mais il est aussi clé étrangère. L'attribut a été nommé plus tôt dans le script comme clé étrangère	**PRIMARY KEY(***Nom_attribut***)** Cet attribut a été nommé plus tôt dans le script comme clé étrangère et doit porter une clause **NOT NULL**
19.	{ENTRE *val*$_1$ et *val*$_2$}		**CONSTRAINT** *Nom_contrainte* **CHECK(***Nom_attribut* **BETWEEN** *val*$_1$ **AND** *val*$_2$**)**

CAS DE MODÉLISATION PHYSIQUE DES DONNÉES

CAS 3-1 SOCIÉTÉ ET DÉPARTEMENT

Traduire le modèle relationnel résultant du cas 2-1 en un modèle physique équivalent.

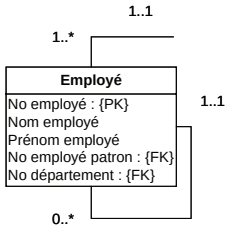


Étape 1.

Table	Exigence	Instruction SQL
Société Nom société : {PK}	1	CREATE TABLE Société ([Nom société] VARCHAR(30) NOT NULL PRIMARY KEY)

Étape 2.

Table	Exigence	Instruction SQL
1..1 Département No département : {PK} Nom société : {FK} 1..*	1 4 3	CREATE TABLE Département ([No département] SMALLINT NOT NULL PRIMARY KEY, [Nom société] VARCHAR(30) NOT NULL REFERENCES Société([Nom société]))

Table	Exigence	Instruction SQL
	 1 4 3 4 3	CREATE TABLE Employé ([No employé] SMALLINT NOT NULL PRIMARY KEY, [Nom employé] VARCHAR(30), [Prénom employé] VARCHAR(30), [No employé patron] SMALLINT NOT NULL REFERENCES Employé([No employé]), [No département] SMALLINT NOT NULL REFERENCES Département([No département]))

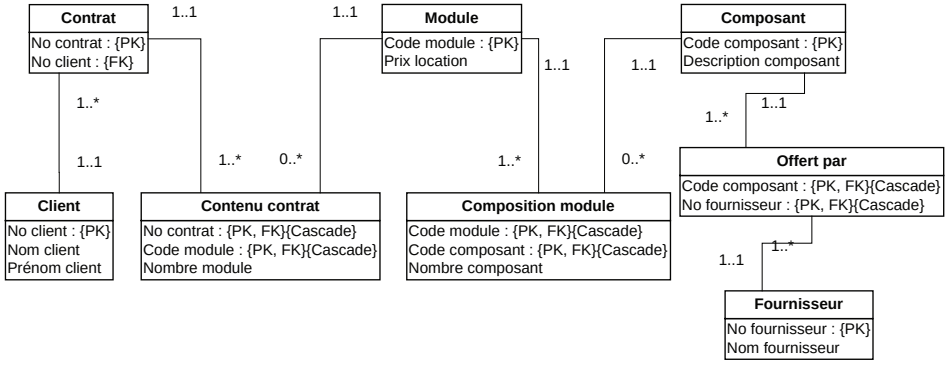
Étape 3.
N/A

Étape 4.
Nombre de tables : 3.

CAS 3-2 LOCATEUR D'ABRIS.

Traduire le modèle relationnel résultant du cas 2-2 en un modèle physique équivalent.

Contraintes de domaine : les attributs **Nombre module** et **Nombre composant** sont strictement supérieurs à 0.
Contraintes d'intégrité : les attributs **Nombre module** et **Nombre composant** sont non nuls.

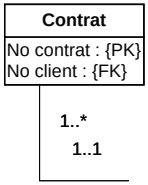


Étape 1.

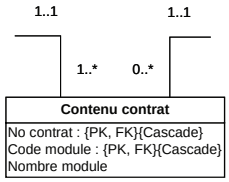
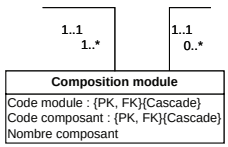
Mise en garde : **Module** est un mot clé du langage SQL. Aussi doit-il apparaître entre crochets dans le script pour éviter toute erreur de syntaxe.

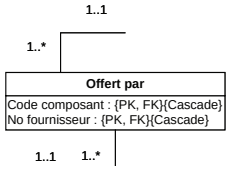
Table	Exigence	Instruction SQL
Client No client : {PK} Nom client Prénom client	1	`CREATE TABLE Client ([No client] SMALLINT NOT NULL PRIMARY KEY, [Nom client] VARCHAR(30), [Prénom client] VARCHAR(30))`
Fournisseur No fournisseur : {PK} Nom fournisseur	1	`CREATE TABLE Fournisseur ([No fournisseur] SMALLINT NOT NULL PRIMARY KEY, [Nom fournisseur] VARCHAR(30))`
Module Code module : {PK} Prix location	1	`CREATE TABLE [Module] ([Code module] VARCHAR(6) NOT NULL PRIMARY KEY, [Prix location] CURRENCY)`
Composant Code composant : {PK} Description composant	1	`CREATE TABLE Composant ([Code composant] VARCHAR(6) NOT NULL PRIMARY KEY, [Description composant] VARCHAR(30))`

Étape 2.

Table	Exigence	Instruction CREATE TABLE
	 1 4 3	CREATE TABLE Contrat ([No contrat] SMALLINT NOT NULL PRIMARY KEY, [No client] SMALLINT NOT NULL REFERENCES Client([No client]))

Étape 3.

Table	Exigence	Instruction SQL
	 14 15 14 15 2 14 7	CREATE TABLE [Contenu contrat] ([No contrat] SMALLINT NOT NULL REFERENCES Contrat([No contrat]) ON DELETE CASCADE ON UPDATE CASCADE, [Code module] VARCHAR(6) NOT NULL REFERENCES [Module]([Code module]) ON DELETE CASCADE ON UPDATE CASCADE, [Nombre module] SMALLINT NOT NULL, PRIMARY KEY([No contrat],[Code module]), CONSTRAINT NombreSupZéro CHECK([Nombre module]>0))
	 14 15 14 15 2 14 7	CREATE TABLE [Composition module] ([Code module] VARCHAR(6) NOT NULL REFERENCES [Module]([Code module]) ON DELETE CASCADE ON UPDATE CASCADE, [Code composant] VARCHAR(6) NOT NULL REFERENCES Composant([Code composant]) ON DELETE CASCADE ON UPDATE CASCADE, [Nombre composant] SMALLINT NOT NULL, PRIMARY KEY([Code module], [Code composant]), CONSTRAINT NombreSupZéro2 CHECK([Nombre composant]>0))



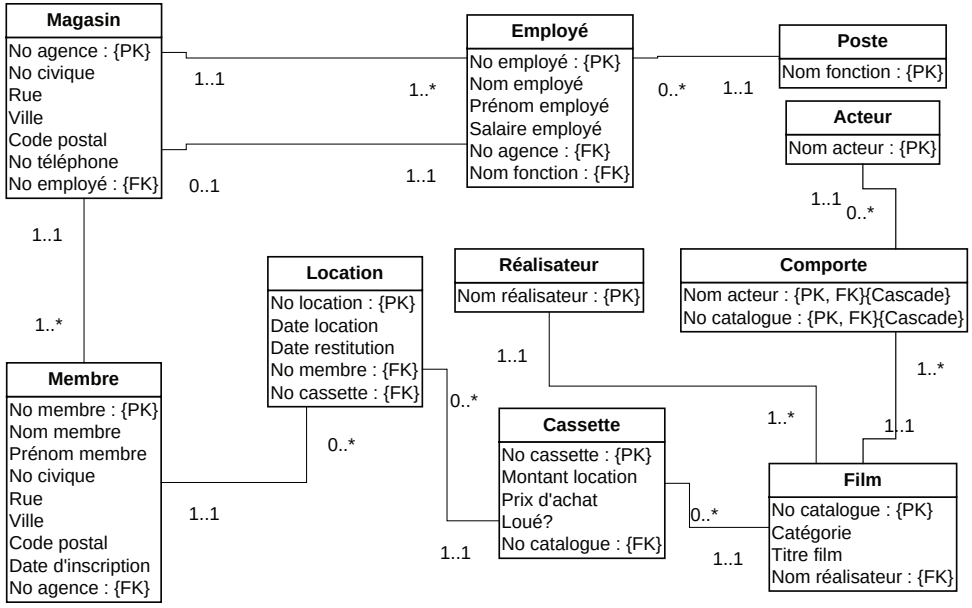

```
                    CREATE TABLE [Offert par]
                     ([Code composant] VARCHAR(6)
14                   NOT NULL
15                   REFERENCES Composant([Code composant])
                     ON DELETE CASCADE
                     ON UPDATE CASCADE,
14                   [No fournisseur] SMALLINT NOT NULL
15                   REFERENCES Fournisseur([No fournisseur])
                     ON DELETE CASCADE
                     ON UPDATE CASCADE,
                     PRIMARY KEY([Code composant],
14                   [No fournisseur]))
```

Étape 4.
Nombre de tables : 8.

CAS 3-3 VIDÉO CLUB

Traduire le modèle relationnel résultant du cas 2-3 en un modèle physique équivalent.

Contraintes de domaine : l'attribut **Salaire employé** est strictement supérieur à 0 ; les valeurs acceptables pour l'attribut **Catégorie** sont 'Horreur', 'Tragédie', 'Comédie', 'Policier'.
Contraintes d'intégrité : **Date location** <= **Date restitution**.

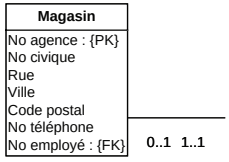
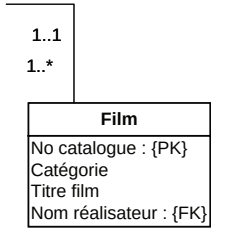
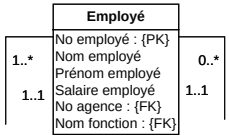


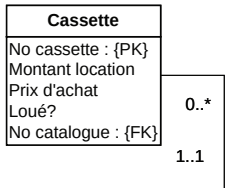
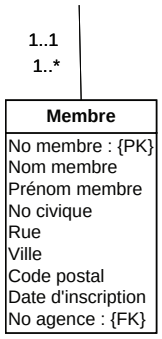
Étape 1.

Table	Exigence	Instruction SQL
Poste Nom fonction : {PK}	1	`CREATE TABLE Poste ([Nom fonction] VARCHAR(30) NOT NULL PRIMARY KEY)`
Acteur Nom acteur : {PK}	1	`CREATE TABLE Acteur ([Nom acteur] VARCHAR(30) NOT NULL PRIMARY KEY)`
Réalisateur Nom réalisateur : {PK}	1	`CREATE TABLE Réalisateur ([Nom réalisateur] VARCHAR(30) NOT NULL PRIMARY KEY)`

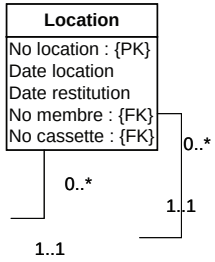
Étape 2.

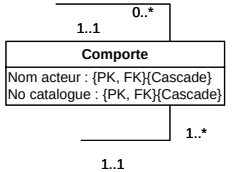
La réalisation de cette étape pose une difficulté. Il existe deux tables qui sont filles l'une de l'autre compte tenu de la présence de deux associations un plusieurs entre elles. Une des tables, **Magasin**, sera créée sans la clé étrangère l'associant à l'autre et à la suite de l'instruction créant **Employé**, une instruction **ALTER TABLE** sera nécessaire pour ajouter la clé étrangère **No employé** à la table **Magasin**. Rappelons que l'on ne peut faire appel à une clause **REFERENCES** portant sur une table qui n'a pas encore été créée.

Table	Exigence	Instruction SQL
	1	CREATE TABLE Magasin ([No agence] SMALLINT NOT NULL PRIMARY KEY, [No civique] VARCHAR(6), [Rue] VARCHAR(30), [Ville] VARCHAR(30), [Code postal] VARCHAR(6))
	1 4 3 10	CREATE TABLE Film ([No catalogue] SMALLINT NOT NULL PRIMARY KEY, [Catégorie] VARCHAR(10), [Titre film] VARCHAR(30), [Nom réalisateur] VARCHAR(30) NOT NULL REFERENCES Réalisateur([Nom réalisateur]), CONSTRAINT CatValide CHECK(Catégorie IN ('Horreur', 'Tragédie', 'Comédie', 'Policier')))
	1 4 3 4 3 7	CREATE TABLE Employé ([No employé] SMALLINT NOT NULL PRIMARY KEY, [Nom employé] VARCHAR(30), [Prénom employé] VARCHAR(30), [Salaire employé] CURRENCY, [No agence] SMALLINT NOT NULL REFERENCES Magasin([No agence]), [Nom fonction] VARCHAR(30) NOT NULL REFERENCES Poste([Nom fonction]), CONSTRAINT SalSupZéro CHECK([Salaire employé]>0))

		ALTER TABLE Magasin ADD COLUMN [No employé] SMALLINT 4 NOT NULL 9 UNIQUE 3 REFERENCES Employé([No employé])
		CREATE TABLE Cassette ([No cassette] SMALLINT 1 NOT NULL PRIMARY KEY, [Montant location] CURRENCY, [Prix d'achat] CURRENCY, [Loué?] BIT, [No catalogue] SMALLINT 4 NOT NULL 3 REFERENCES Film([No catalogue]))
		CREATE TABLE Membre ([No membre] SMALLINT 1 NOT NULL PRIMARY KEY, [Nom membre] VARCHAR(30), [Prénom membre] VARCHAR(30), [No civique] VARCHAR(6), [Rue] VARCHAR(30), [Ville] VARCHAR(30), [Code postal] VARCHAR(6), [Date d'inscription] DATE, [No agence] SMALLINT 4 NOT NULL 3 REFERENCES Magasin([No agence]))

Étape 3.

Table	Exigence	Instruction SQL
		CREATE TABLE [Location] ([No location] SMALLINT 1 NOT NULL PRIMARY KEY, [Date location] DATE, [Date restitution] DATE, [No membre] SMALLINT 4 NOT NULL 3 REFERENCES Membre([No membre]), [No cassette] SMALLINT 4 NOT NULL 3 REFERENCES Cassette([No cassette]), 12 CONSTRAINT LocationInfRestitution CHECK([Date location]<= [Date restitution]))



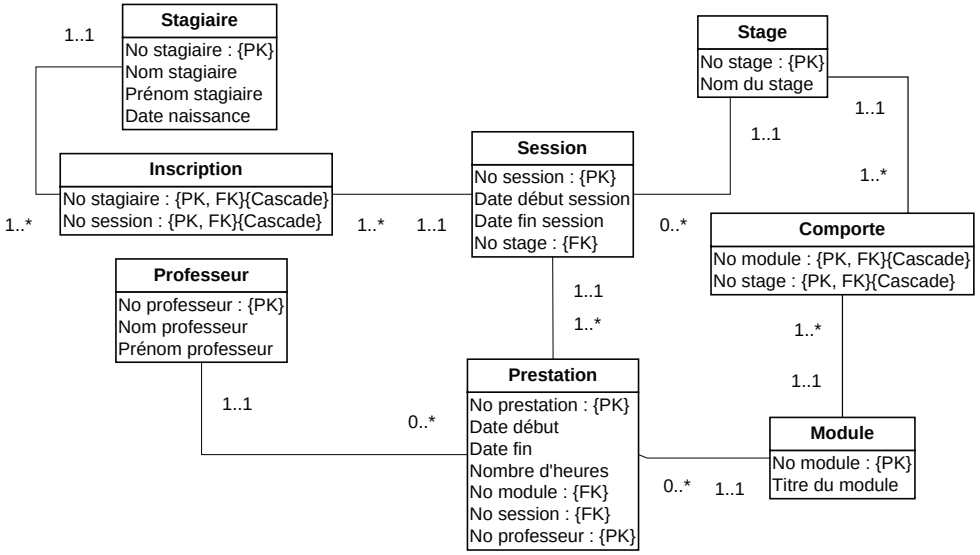

```
                CREATE TABLE [Comporte]
14               ([Nom acteur] VARCHAR(30) NOT NULL
15               REFERENCES Acteur([Nom acteur])
                 ON DELETE CASCADE
                 ON UPDATE CASCADE,
14               [No catalogue] SMALLINT NOT NULL
15               REFERENCES Film([No catalogue])
                 ON DELETE CASCADE
                 ON UPDATE CASCADE,
14               PRIMARY KEY([Nom acteur],
                 [No catalogue]))
```

Étape 4.
Nombre de tables : 10.

CAS 3-4 GESTION DE STAGE

Traduire le modèle relationnel résultant du cas 2-4 en un modèle physique équivalent.

Contraintes de domaine : l'attribut **Nombre d'heures** possède une valeur par défaut = 1.
Contraintes d'intégrité : l'attribut **Nombre d'heures** est strictement supérieur à 1.



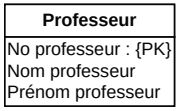
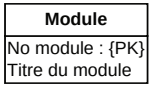
Étape 1.

Mise en garde : **Module** et **Session** sont des mots clés du langage SQL. Aussi doivent-ils apparaître entre crochets dans le script pour éviter toute erreur de syntaxe.

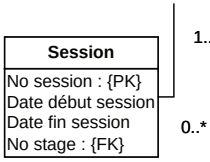
Table	Exigence	Instruction SQL

Stagiaire
No stagiaire : {PK}
Nom stagiaire
Prénom stagiaire
Date naissance
	1	`CREATE TABLE Stagiaire`
		`([No stagiaire] SMALLINT`
		`NOT NULL PRIMARY KEY,`
		`[Nom stagiaire] VARCHAR(30),`
		`[Prénom stagiaire] VARCHAR(30),`
		`[Date naissance] DATE)`

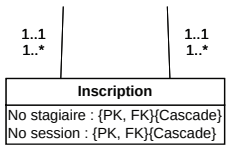
Stage
No stage : {PK}
Nom du stage
	1	`CREATE TABLE Stage`
		`([No stage] SMALLINT`
		`NOT NULL PRIMARY KEY,`
		`[Nom du stage] VARCHAR(30))`

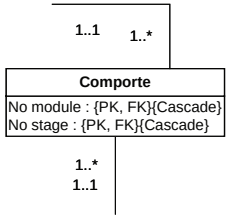
	1	CREATE TABLE Professeur ([No professeur] SMALLINT NOT NULL PRIMARY KEY, [Nom professeur] VARCHAR(30), [Prénom professeur] VARCHAR(30))
		CREATE TABLE [Module] ([No module] SMALLINT NOT NULL PRIMARY KEY, [Titre du module] VARCHAR(30))

Étape 2.

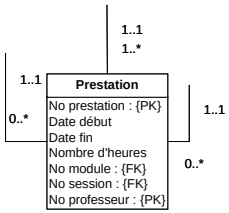
Table	Exigence	Instruction SQL
	1 4 3	CREATE TABLE [Session] ([No session] SMALLINT NOT NULL PRIMARY KEY, [Date début session] DATE, [Date fin session] DATE, [No stage] SMALLINT NOT NULL REFERENCES Stage([No stage]))

Étape 3.

Table	Exigence	Instruction SQL
	14 15 14 15 14	CREATE TABLE [Inscription] ([No stagiaire] SMALLINT NOT NULL REFERENCES Stagiaire([No stagiaire]) ON DELETE CASCADE ON UPDATE CASCADE, [No session] SMALLINT NOT NULL REFERENCES [Session]([No session]) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY([No stagiaire], [No session]))




```
CREATE TABLE [Comporte]
 ([No module] SMALLINT NOT NULL
 REFERENCES [Module]([No module])
 ON DELETE CASCADE
 ON UPDATE CASCADE,
 [No stage] SMALLINT NOT NULL
 REFERENCES Stage([No stage])
 ON DELETE CASCADE
 ON UPDATE CASCADE,
 PRIMARY KEY([No module],
 [No stage]))
```

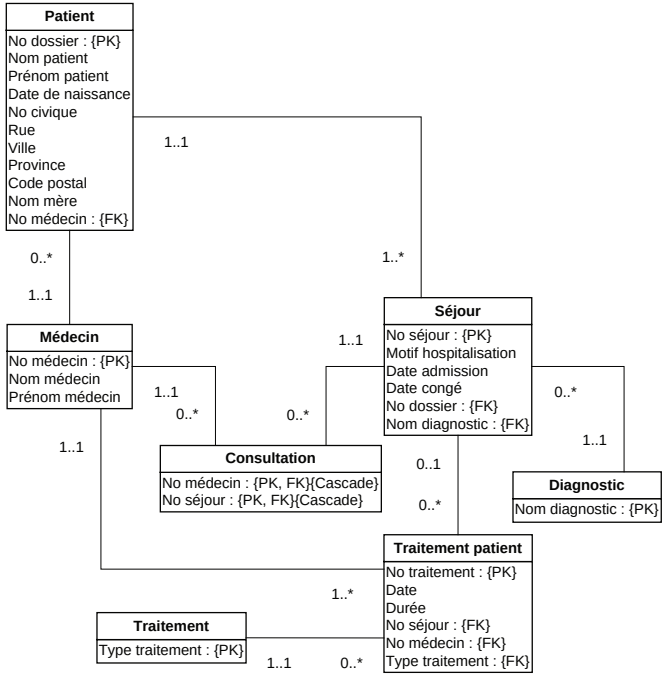



```
CREATE TABLE Prestation
 ([No prestation] SMALLINT
 NOT NULL PRIMARY KEY,
 [Date début] DATE,
 [Date fin] DATE,
 [Nombre d'heures] SMALLINT
 DEFAULT 1,
 [No module] SMALLINT
 NOT NULL
 REFERENCES [Module]([No module]),
 [No session] SMALLINT
 NOT NULL
 REFERENCES [Session]([No session]),
 [No professeur] SMALLINT
 NOT NULL
 REFERENCES Professeur([No professeur]),
 CONSTRAINT HeuresSupUn
 CHECK([Nombre d'heures]>1))
```

Étape 4.
Nombre de tables : 8.

CAS 3-5 DOSSIER PATIENT

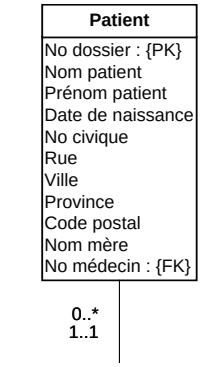
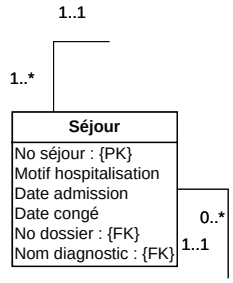
Traduire le modèle relationnel résultant du cas 2-5 en un modèle physique équivalent.



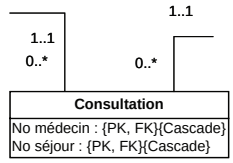
Étape 1.

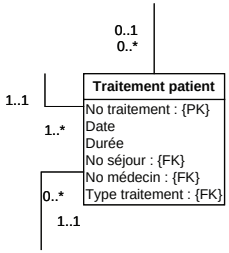
Table	Exigence	Instruction SQL
Médecin No médecin : {PK} Nom médecin Prénom médecin	1	`CREATE TABLE Médecin ([No médecin] SMALLINT NOT NULL PRIMARY KEY, [Nom médecin] VARCHAR(30), [Prénom médecin] VARCHAR(30))`
Diagnostic Nom diagnostic : {PK}	1	`CREATE TABLE Diagnostic ([Nom diagnostic] VARCHAR(30) NOT NULL PRIMARY KEY)`
Traitement Type traitement : {PK}	1	`CREATE TABLE Traitement ([Type traitement] VARCHAR(30) NOT NULL PRIMARY KEY)`

Étape 2.

Table	Exigence	Instruction SQL
	 1 4 3	CREATE TABLE Patient ([No dossier] SMALLINT NOT NULL PRIMARY KEY, [Nom patient] VARCHAR(30), [Prénom patient] VARCHAR(30), [Date de naissance] DATE, [No civique] VARCHAR(6), [Rue] VARCHAR(30), [Ville] VARCHAR(30), [Province] VARCHAR(30), [Code postal] VARCHAR(6), [Nom mère] VARCHAR(30), [No médecin] SMALLINT NOT NULL REFERENCES Médecin([No médecin]))
	 1 4 3 4 3	CREATE TABLE Séjour ([No séjour] SMALLINT NOT NULL PRIMARY KEY, [Motif hospitalisation] VARCHAR(30), [Date admission] DATE, [Date congé] DATE, [No dossier] SMALLINT NOT NULL REFERENCES Patient([No dossier]), [Nom diagnostic] VARCHAR(30) NOT NULL REFERENCES Diagnostic([Nom diagnostic]))

Étape 3.

Table	Exigence	Instruction SQL
	 14 15 14 15 14	CREATE TABLE [Consultation] ([No médecin] SMALLINT NOT NULL REFERENCES Médecin([No médecin]) ON DELETE CASCADE ON UPDATE CASCADE, [No séjour] SMALLINT NOT NULL REFERENCES Séjour([No séjour]) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY([No médecin],[No séjour]))

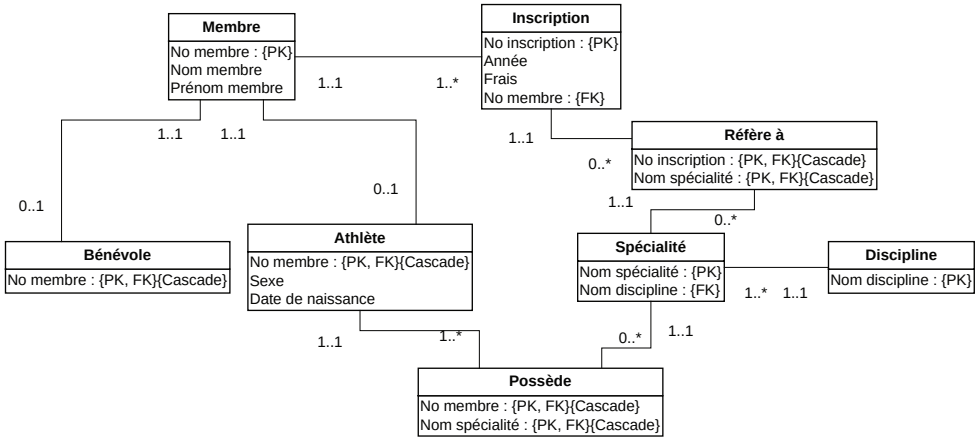



```
CREATE TABLE [Traitement patient]
 ([No traitement] SMALLINT
1 NOT NULL PRIMARY KEY,
  [Date] DATE,
  [Durée] FLOAT,
  [No séjour] SMALLINT
4 NOT NULL
3 REFERENCES Séjour([No séjour]),
  [No médecin] SMALLINT
4 NOT NULL
3 REFERENCES Médecin([No médecin]),
  [Type traitement] VARCHAR(30)
4 NOT NULL
3 REFERENCES Traitement([Type traitement]))
```

Étape 4.
Nombre de tables : 7.

CAS 3-6 CLUB OLYMPIQUE

Traduire le modèle relationnel résultant du cas 2-8 en un modèle physique équivalent.

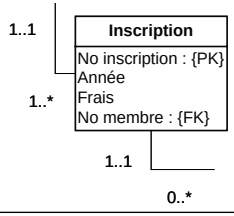
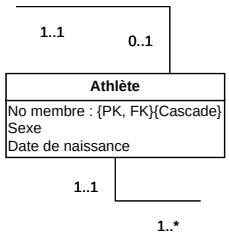


Étape 1.

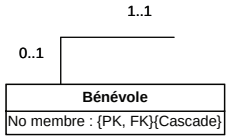
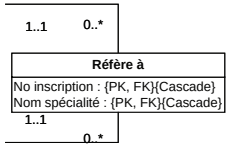
Table	Exigence	Instruction SQL
Membre No membre : {PK} Nom membre Prénom membre	1	`CREATE TABLE Membre ([No membre] SMALLINT NOT NULL PRIMARY KEY, [Nom membre] VARCHAR(30), [Prénom membre] VARCHAR(30))`
Discipline Nom discipline : {PK}	1	`CREATE TABLE Discipline ([Nom discipline] VARCHAR(30) NOT NULL PRIMARY KEY)`

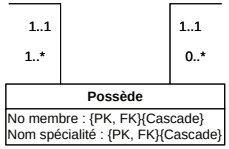
Étape 2.

Table	Exigence	Instruction SQL
0..* 1..1 **Spécialité** Nom spécialité : {PK} Nom discipline : {FK} 1..1 0..*	1 4 3	`CREATE TABLE Spécialité ([Nom spécialité] VARCHAR(30) NOT NULL PRIMARY KEY, [Nom discipline] VARCHAR(30) NOT NULL REFERENCES Discipline([Nom discipline]))`

	1 4 3	CREATE TABLE Inscription ([No inscription] SMALLINT NOT NULL PRIMARY KEY, [Année] SMALLINT, [Frais] CURRENCY, [No membre] SMALLINT NOT NULL REFERENCES Membre([No membre]))
	18,9 15 18	CREATE TABLE [Athlète] ([No membre] SMALLINT NOT NULL UNIQUE REFERENCES Membre([No membre]) ON DELETE CASCADE ON UPDATE CASCADE, [Sexe] VARCHAR(1), [Date de naissance] DATE, PRIMARY KEY([No membre]))

Étape 3.

Table	Exigence	Instruction SQL
	18,9 15 18	CREATE TABLE [Bénévole] ([No membre] SMALLINT NOT NULL UNIQUE REFERENCES Membre([No membre]) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY([No membre]))
	14 15 14 15 14	CREATE TABLE [Réfère à] ([No inscription] SMALLINT NOT NULL REFERENCES Inscription([No inscription]) ON DELETE CASCADE ON UPDATE CASCADE, [Nom spécialité] VARCHAR(30) NOT NULL REFERENCES Spécialité([Nom spécialité]) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY([No inscription],[Nom spécialité]))

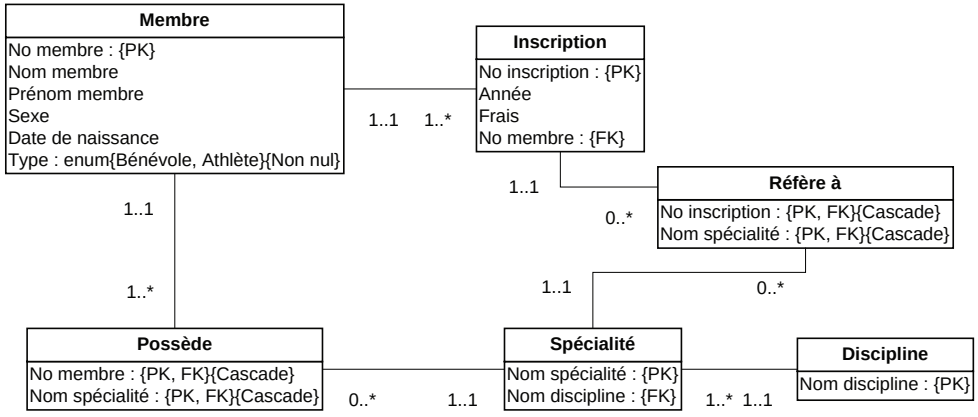



```
                CREATE TABLE [Possède]
14               ([No membre] SMALLINT NOT NULL
15               REFERENCES Athlète([No membre])
                 ON DELETE CASCADE
                 ON UPDATE CASCADE,
14               [Nom spécialité] VARCHAR(30) NOT NULL
15               REFERENCES Spécialité([Nom spécialité])
                 ON DELETE CASCADE
                 ON UPDATE CASCADE,
14               PRIMARY KEY([No membre],[Nom spécialité]))
```

Étape 4.
Nombre de tables : 8.

CAS 3-7 CLUB OLYMPIQUE, VERSION PARTITION

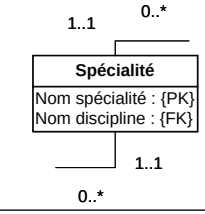
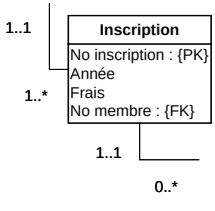
Traduire le modèle relationnel résultant du cas 2-9 en un modèle physique équivalent.



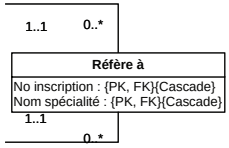
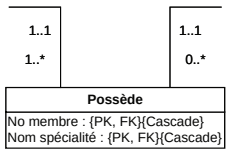
Étape 1.

Table	Exigence	Instruction SQL
Membre No membre : {PK} Nom membre Prénom membre Sexe Date de naissance Type : enum{Bénévole, Athlète}{Non nul}	1 2 10	CREATE TABLE Membre ([No membre] SMALLINT NOT NULL PRIMARY KEY, [Nom membre] VARCHAR(30), [Prénom membre] VARCHAR(30), [Sexe] VARCHAR(1), [Date de naissance] DATE, [Type] VARCHAR(10) NOT NULL, CONSTRAINT TypeValide CHECK([Type] IN ('Bénévole', 'Athlète')))
Discipline Nom discipline : {PK}	1	CREATE TABLE Discipline ([Nom discipline] VARCHAR(30) NOT NULL PRIMARY KEY)

Étape 2.

Table	Exigence	Instruction SQL
	 1 4 3	CREATE TABLE Spécialité ([Nom spécialité] VARCHAR(30) NOT NULL PRIMARY KEY, [Nom discipline] VARCHAR(30) NOT NULL REFERENCES Discipline([Nom discipline]))
	 1 4 3	CREATE TABLE Inscription ([No inscription] SMALLINT NOT NULL PRIMARY KEY, [Année] SMALLINT, [Frais] CURRENCY, [No membre] SMALLINT NOT NULL REFERENCES Membre([No membre]))

Étape 3.

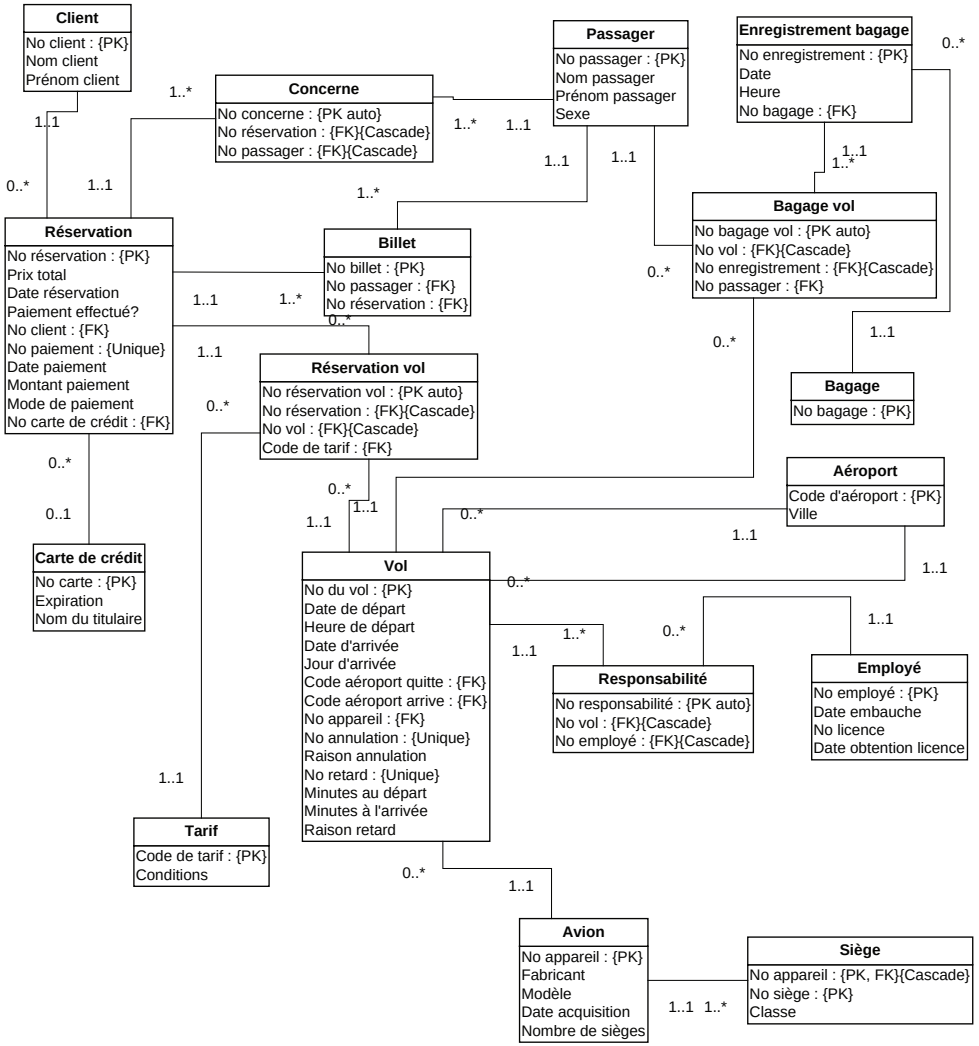
Table	Exigence	Instruction SQL
	 14 15 14 15 14	CREATE TABLE [Réfère à] ([No inscription] SMALLINT NOT NULL REFERENCES Inscription([No inscription]) ON DELETE CASCADE ON UPDATE CASCADE, [Nom spécialité] VARCHAR(30) NOT NULL REFERENCES Spécialité([Nom spécialité]) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY([No inscription],[Nom spécialité]))
	 14 15 14 15 14	CREATE TABLE [Possède] ([No membre] SMALLINT NOT NULL REFERENCES Membre([No membre]) ON DELETE CASCADE ON UPDATE CASCADE, [Nom spécialité] VARCHAR(30) NOT NULL REFERENCES Spécialité([Nom spécialité]) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY([No membre],[Nom spécialité]))

Étape 4.

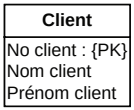
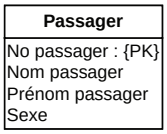
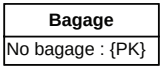
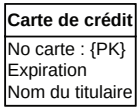
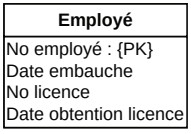
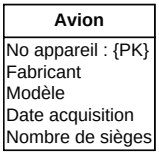
Nombre de tables : 6.

CAS 3-8 TRANSPORTEUR AÉRIEN, VERSION TOTALEMENT OPTIMISÉE (FIGURE 2-40)

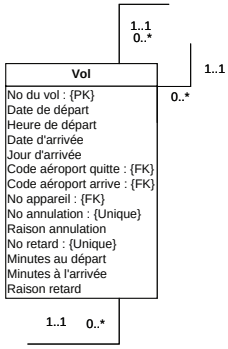
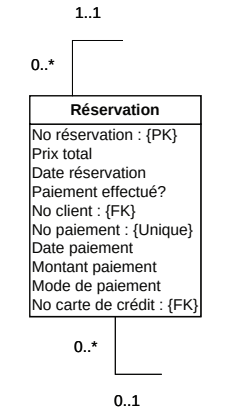
Traduire le modèle relationnel de la figure 2-40 en un modèle physique équivalent.

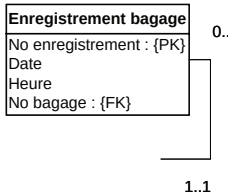


Étape 1.

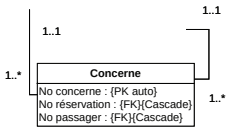
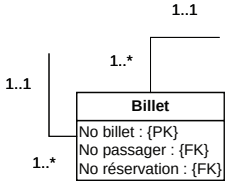
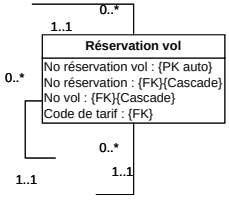
Table	Exigence	Instruction SQL
	1	`CREATE TABLE Client ([No client] SMALLINT NOT NULL PRIMARY KEY, [Nom client] VARCHAR(30), [Prénom client] VARCHAR(30))`
	1	`CREATE TABLE Passager ([No passager] SMALLINT NOT NULL PRIMARY KEY, [Nom passager] VARCHAR(30), [Prénom passager] VARCHAR(30), [Sexe] VARCHAR(1))`
	1	`CREATE TABLE Bagage ([No bagage] SMALLINT NOT NULL PRIMARY KEY)`
	1	`CREATE TABLE Aéroport ([Code d'aéroport] VARCHAR(3) NOT NULL PRIMARY KEY, [Ville] VARCHAR(30))`
	1	`CREATE TABLE [Carte de crédit] ([No carte] VARCHAR(15) NOT NULL PRIMARY KEY, [Expiration] DATE, [Nom du titulaire] VARCHAR(30))`
	1	`CREATE TABLE Employé ([No employé] SMALLINT NOT NULL PRIMARY KEY, [Date embauche] DATE, [No licence] VARCHAR(10), [Date obtention licence] DATE)`
	1	`CREATE TABLE Tarif ([Code de tarif] VARCHAR(10) NOT NULL PRIMARY KEY, [Conditions] VARCHAR(30))`
	1	`CREATE TABLE Avion ([No appareil] SMALLINT NOT NULL PRIMARY KEY, [Fabricant] VARCHAR(30), [Modèle] VARCHAR(10), [Date acquisition] DATE, [Nombre de sièges] SMALLINT)`

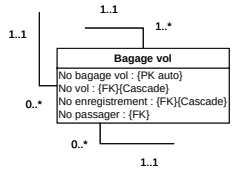
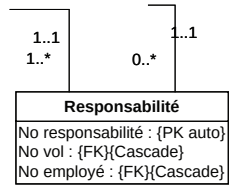
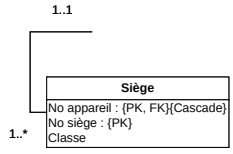
Étape 2.

Table	Exigence	Instruction SQL
	 1 4 3 4 3 4 3 17 17	CREATE TABLE Vol ([No du vol] SMALLINT NOT NULL PRIMARY KEY, [Date de départ] DATE, [Heure de départ] DATE, [Date d'arrivée] DATE, [Jour d'arrivée] VARCHAR(10), [Code aéroport quitte] VARCHAR(3) NOT NULL REFERENCES Aéroport([Code d'aéroport]), [Code aéroport arrive] VARCHAR(3) NOT NULL REFERENCES Aéroport([Code d'aéroport]), [No appareil] SMALLINT NOT NULL REFERENCES Avion([No appareil]), [No annulation] SMALLINT UNIQUE, [Raison annulation] VARCHAR(30), [No retard] SMALLINT UNIQUE, [Minutes au départ] SMALLINT, [Minutes à l'arrivée] SMALLINT, [Raison retard] VARCHAR(30))
	 1 4 3 17 9 4 3	CREATE TABLE Réservation ([No réservation] SMALLINT NOT NULL PRIMARY KEY, [Prix total] CURRENCY, [Date réservation] DATE, [Paiement effectué] BIT, [No client] SMALLINT NOT NULL REFERENCES Client([No client]), [No paiement] SMALLINT UNIQUE, [Date paiement] DATE, [Montant paiement] CURRENCY, [No carte de crédit] VARCHAR(15) UNIQUE NOT NULL REFERENCES [Carte de crédit]([No carte]))

	1 4 3	CREATE TABLE [Enregistrement bagage] ([No enregistrement] SMALLINT NOT NULL PRIMARY KEY, [Date] DATE, [Heure] DATE, [No bagage] SMALLINT NOT NULL REFERENCES Bagage([No bagage]))

Étape 3.

Table	Exigence	Instruction SQL
	11 14 15 14 15	CREATE TABLE Concerne ([No concerne] IDENTITY NOT NULL PRIMARY KEY, [No réservation] SMALLINT NOT NULL REFERENCES Réservation([No réservation]) ON DELETE CASCADE ON UPDATE CASCADE, [No passager] SMALLINT NOT NULL REFERENCES Passager([No passager]) ON DELETE CASCADE ON UPDATE CASCADE)
	1 4 3 4 3	CREATE TABLE [Billet] ([No billet] SMALLINT NOT NULL PRIMARY KEY, [No passager] SMALLINT NOT NULL REFERENCES Passager([No passager]), [No réservation] SMALLINT NOT NULL REFERENCES Réservation([No réservation]))
	11 4 15 4 15 4 3	CREATE TABLE [Réservation vol] ([No réservation vol] IDENTITY NOT NULL PRIMARY KEY, [No réservation] SMALLINT NOT NULL REFERENCES Réservation([No réservation]) ON DELETE CASCADE ON UPDATE CASCADE, [No vol] SMALLINT NOT NULL REFERENCES Vol([No du vol]) ON DELETE CASCADE ON UPDATE CASCADE, [Code de tarif] VARCHAR(10) NOT NULL REFERENCES Tarif([Code de tarif]))

	11 4 15 4 15 4 3	CREATE TABLE [Bagage vol] ([No bagage vol] IDENTITY NOT NULL PRIMARY KEY, [No vol] SMALLINT NOT NULL REFERENCES Vol([No du vol]) ON DELETE CASCADE ON UPDATE CASCADE, [No enregistrement] SMALLINT NOT NULL REFERENCES [Enregistrement bagage]([No enregistrement]) ON DELETE CASCADE ON UPDATE CASCADE, [No passager] SMALLINT NOT NULL REFERENCES Passager([No passager]))
	11 4 15 4 15	CREATE TABLE [Responsabilité] ([No responsabilité] IDENTITY NOT NULL PRIMARY KEY, [No vol] SMALLINT NOT NULL REFERENCES Vol([No du vol]) ON DELETE CASCADE ON UPDATE CASCADE, [No employé] SMALLINT NOT NULL REFERENCES Employé([No employé]) ON DELETE CASCADE ON UPDATE CASCADE)
	14 15 14	CREATE TABLE [Siège] ([No appareil] SMALLINT NOT NULL REFERENCES Avion([No appareil]) ON DELETE CASCADE ON UPDATE CASCADE, [No siège] SMALLINT, [Classe] VARCHAR(10), PRIMARY KEY([No appareil],[No siège]))

Étape 4.
Nombre de tables : 17.

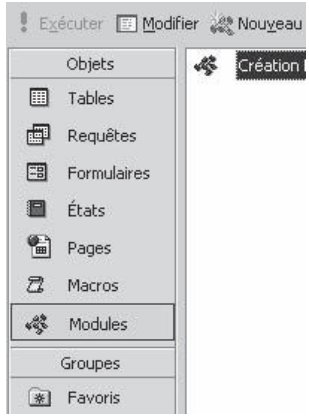
RÉALISATION DU MODÈLE PHYSIQUE EN SQL AVEC MS ACCESS

MS Access dispose, comme la plupart des SGBD, d'un langage de programmation pour la réalisation d'applications de bases de données. Il porte le nom *Visual Basic For Applications*, ou plus simplement VBA, langage par ailleurs intégré à bon nombre de logiciels de la suite MS Office dont Excel et Word.

VBA incorpore, par le biais de la bibliothèque de programmes ADO, des mécanismes pour l'exécution de requêtes de définition de données tels que **`CREATE TABLE`** ou **`ALTER TABLE`**. Pour réaliser un script de création de tables en MS Access, il suffit de créer un module Access et d'y incorporer une procédure regroupant des instructions permettant d'exécuter toutes les requêtes de création de tables du modèle physique.

La figure 3-6 montre une partie de la fenêtre qui permet de créer, modifier et exécuter des objets d'une BD en MS Access. La création d'un module s'effectue en sélectionnant le bouton *Modules* et en cliquant ensuite sur le bouton *Nouveau* situé au haut de la fenêtre.

FIGURE 3-6 **Création d'un module en MS Access**



À l'ouverture de la fenêtre contenant l'instruction :
`Option Compare Database`,
il importe d'ajouter à sa suite une procédure appelée par exemple :
`Script_Creation_Tables()`
et possédant les instructions suivantes. Les trois apostrophes au centre de la procédure devront être remplacées par une suite d'instructions dédiées à la création des tables.

```
Private Sub Script_Creation_Tables()
Dim conDatabase As ADODB.Connection
Dim SQL As String
On Error GoTo Error_Handler
Set conDatabase = Application.CurrentProject.Connection

'
'
'
conDatabase.Close
Set conDatabase = Nothing
Exit Sub
Error_Handler:
 MsgBox Err.Description, vbInformation
End Sub
```

Chaque bloc d'instructions conçu pour créer une table appelée **Client** aura un contenu similaire au bloc donné ci-dessous.

```
SQL = _
"CREATE TABLE Client" & _
" ([No client] SMALLINT" & _
" NOT NULL PRIMARY KEY," & _
" [Nom client] VARCHAR(30)," & _
" [Prénom client] VARCHAR(30))"

conDatabase.Execute SQL
MsgBox "Table Client créée", vbInformation
```

SQL est une variable déclarée dans la procédure du script de création de la table. Le caractère souligné _ , indique que la ligne suivante contient la suite de l'instruction d'affectation. Les éléments du script sont ici disposés sur plusieurs lignes. L'opérateur de concaténation **&** est utilisé pour indiquer que le contenu de chaque élément placé entre guillemets doit être combiné à celui de la ligne suivante.

L'instruction **conDatabase.Execute SQL** exécute l'instruction stockée dans la variable **SQL**. Elle procède donc à la création de la table dans la BD.

La dernière instruction fait apparaître à l'écran une boîte de dialogue informant l'utilisateur que la table a été créée:

```
MsgBox "Table Client créée", vbInformation
```

La procédure finale doit comporter autant de blocs d'instructions de cette nature qu'il y a de tables à créer dans la BD. Nous donnons ci-après le contenu complet de la procédure requise pour créer les tables du modèle physique produit dans le cadre du cas 3-8. Pour lancer l'exécution de la procédure, il suffit de placer le pointeur au début de la procédure et de cliquer ensuite sur le bouton déclenchant l'exécution d'une procédure. Le bouton d'exécution ▸ est placé en haut de la fenêtre dans la barre d'outils.

```
Private Sub Script_Creation_Tables()
Dim conDatabase As ADODB.Connection
Dim SQL As String

On Error GoTo Error_Handler
Set conDatabase = Application.CurrentProject.Connection

SQL = _
"CREATE TABLE Client" & _
" ([No client] SMALLINT" & _
" NOT NULL PRIMARY KEY," & _
" [Nom client] VARCHAR(30)," & _
" [Prénom client] VARCHAR(30))"

conDatabase.Execute SQL
MsgBox "Table Client créée", vbInformation

SQL = _
"CREATE TABLE Passager" & _
" ([No passager] SMALLINT" & _
" NOT NULL PRIMARY KEY," & _
" [Nom passager] VARCHAR(30)," & _
" [Prénom passager] VARCHAR(30)," & _
" [Sexe] VARCHAR(1))"

conDatabase.Execute SQL
MsgBox "Table Passager créée", vbInformation

SQL = _
"CREATE TABLE Bagage" & _
" ([No bagage] SMALLINT" & _
" NOT NULL PRIMARY KEY)"

conDatabase.Execute SQL
```

```
MsgBox "Table Bagage créée", vbInformation

SQL = _
"CREATE TABLE Aéroport" & _
" ([Code d'aéroport] VARCHAR(3)" & _
" NOT NULL PRIMARY KEY," & _
" [Ville] VARCHAR(30))"

conDatabase.Execute SQL
MsgBox "Table Aéroport créée", vbInformation

SQL = _
"CREATE TABLE [Carte de crédit]" & _
" ([No carte] VARCHAR(15)" & _
" NOT NULL PRIMARY KEY," & _
" [Expiration] DATE," & _
" [Nom du titulaire] VARCHAR(30))"

conDatabase.Execute SQL
MsgBox "Table Carte de crédit créée", vbInformation

SQL = _
"CREATE TABLE Employé" & _
" ([No employé] SMALLINT" & _
" NOT NULL PRIMARY KEY," & _
" [Date embauche] DATE," & _
" [No licence] VARCHAR(10)," & _
" [Date obtention licence] DATE)"

conDatabase.Execute SQL
MsgBox "Table Employé créée", vbInformation

SQL = _
"CREATE TABLE Tarif" & _
" ([Code de tarif] VARCHAR(10)" & _
" NOT NULL PRIMARY KEY," & _
" [Conditions] VARCHAR(30))"

conDatabase.Execute SQL
MsgBox "Table Tarif créée", vbInformation

SQL = _
```

```
"CREATE TABLE Avion" & _
" ([No appareil] SMALLINT" & _
" NOT NULL PRIMARY KEY," & _
" [Fabricant] VARCHAR(30)," & _
" [Modèle] VARCHAR(10)," & _
" [Date acquisition] DATE," & _
" [Nombre de sièges] SMALLINT)"

conDatabase.Execute SQL
MsgBox "Table Avion créée", vbInformation

SQL = _
"CREATE TABLE Vol" & _
" ([No du vol] SMALLINT " & _
" NOT NULL PRIMARY KEY," & _
" [Date de départ] DATE," & _
" [Heure de départ] DATE," & _
" [Date d'arrivée] DATE," & _
" [Jour d'arrivée] VARCHAR(10)," & _
" [Code aéroport quitte] VARCHAR(3)" & _
" NOT NULL" & _
" REFERENCES Aéroport([Code d'aéroport])," & _
" [Code aéroport arrive] VARCHAR(3)" & _
" NOT NULL" & _
" REFERENCES Aéroport([Code d'aéroport])," & _
" [No appareil] SMALLINT" & _
" NOT NULL" & _
" REFERENCES Avion([No appareil])," & _
" [No annulation] SMALLINT UNIQUE," & _
" [Raison annulation] VARCHAR(30)," & _
" [No retard] SMALLINT UNIQUE," & _
" [Minutes au départ] SMALLINT," & _
" [Minutes à l'arrivée] SMALLINT," & _
" [Raison retard] VARCHAR(30))"

conDatabase.Execute SQL
MsgBox "Table Vol créée", vbInformation

SQL = _
"CREATE TABLE Réservation" & _
" ([No réservation] SMALLINT " & _
" NOT NULL PRIMARY KEY," & _
```

```
" NOT NULL PRIMARY KEY," & _
" [Prix total] CURRENCY, " & _
" [Date réservation] DATE," & _
" [Paiement effectué] BIT," & _
" [No client] SMALLINT" & _
" NOT NULL" & _
" REFERENCES Client([No client])," & _
" [No paiement] SMALLINT UNIQUE," & _
" [Date paiement] DATE," & _
" [Montant paiement] CURRENCY," & _
" [No carte de crédit] VARCHAR(15) " & _
" UNIQUE" & _
" NOT NULL" & _
" REFERENCES [Carte de crédit]([No carte]))"

conDatabase.Execute SQL
MsgBox "Table Réservation créée", vbInformation

SQL = _
"CREATE TABLE [Enregistrement bagage]" & _
" ([No enregistrement] SMALLINT " & _
" NOT NULL PRIMARY KEY, " & _
" [Date] DATE," & _
" [Heure] DATE," & _
" [No bagage] SMALLINT" & _
" NOT NULL" & _
" REFERENCES Bagage([No bagage]))"

conDatabase.Execute SQL
MsgBox "Table Enregistrement bagage créée", vbInformation

SQL = _
"CREATE TABLE Concerne" & _
" ([No concerne] IDENTITY " & _
" NOT NULL PRIMARY KEY," & _
" [No réservation] SMALLINT NOT NULL" & _
" REFERENCES Réservation([No réservation])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [No passager] SMALLINT NOT NULL" & _
" REFERENCES Passager([No passager])" & _
" ON DELETE CASCADE" & _
```

```
" ON UPDATE CASCADE)"

conDatabase.Execute SQL
MsgBox "Table Concerne créée", vbInformation

SQL = _
"CREATE TABLE [Billet]" & _
" ([No billet] SMALLINT " & _
" NOT NULL PRIMARY KEY," & _
" [No passager] SMALLINT " & _
" NOT NULL" & _
" REFERENCES Passager([No passager])," & _
" [No réservation] SMALLINT " & _
" NOT NULL" & _
" REFERENCES Réservation([No réservation]))"

conDatabase.Execute SQL
MsgBox "Table Billet créée", vbInformation

SQL = _
"CREATE TABLE [Réservation vol]" & _
" ([No réservation vol] IDENTITY " & _
" NOT NULL PRIMARY KEY," & _
" [No réservation] SMALLINT NOT NULL" & _
" REFERENCES Réservation([No réservation])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [No vol] SMALLINT NOT NULL" & _
" REFERENCES Vol([No du vol])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [Code de tarif] VARCHAR(10)" & _
" NOT NULL" & _
" REFERENCES Tarif([Code de tarif]))"

conDatabase.Execute SQL
MsgBox "Table Réservation vol créée", vbInformation

SQL = _
"CREATE TABLE [Bagage vol] " & _
" ([No bagage vol] IDENTITY " & _
" NOT NULL PRIMARY KEY," & _
```

```
" [No vol] SMALLINT " & _
" NOT NULL" & _
" REFERENCES Vol([No du vol])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [No enregistrement] SMALLINT " & _
" NOT NULL" & _
" REFERENCES [Enregistrement bagage]([No
enregistrement])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [No passager] SMALLINT" & _
" NOT NULL" & _
" REFERENCES Passager([No passager])) "

conDatabase.Execute SQL
MsgBox "Table Billet créée", vbInformation

SQL = _
"CREATE TABLE [Responsabilité]" & _
" ([No responsabilité] IDENTITY " & _
" NOT NULL PRIMARY KEY," & _
" [No vol] SMALLINT NOT NULL" & _
" REFERENCES Vol([No du vol])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [No employé] SMALLINT NOT NULL" & _
" REFERENCES Employé([No employé])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE)"

conDatabase.Execute SQL
MsgBox "Table Respo créée", vbInformation

SQL = _
"CREATE TABLE [Siège]" & _
" ([No appareil] SMALLINT NOT NULL" & _
" REFERENCES Avion([No appareil])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [No siège] SMALLINT," & _
" [Classe] VARCHAR(10)," & _
```

```
" PRIMARY KEY([No appareil],[No siège]))"

conDatabase.Execute SQL
MsgBox "Table Siège créée", vbInformation

conDatabase.Close
Set conDatabase = Nothing

Exit Sub
Error_Handler:
 MsgBox Err.Description, vbInformation
End Sub
```

Après l'exécution de la procédure, on doit s'assurer d'enregistrer le module en lui donnant un nom pertinent.

La réalisation d'un modèle physique de données en faisant appel au langage SQL peut être lourde, surtout si le modèle comporte plusieurs tables. De plus, elle est sujette à de nombreuses erreurs qu'il peut être fort difficile de repérer :

1. erreurs de syntaxe : virgule manquante, mot clé incorrectement orthographié, parenthèse manquante, clause **`CHECK`** incorrecte, etc.
2. erreurs de logique : création d'une table avec une clé étrangère référant à une table non créée préalablement, type de données de la clé étrangère incompatible avec le type de la clé de la table mère, clé simple créée sur une colonne alors qu'elle devrait être composée, etc.

MS Access offre un support fort limité pour localiser les erreurs de syntaxe d'une instruction **`CREATE TABLE`**. Un message apparaît à l'écran lorsqu'une erreur est détectée. Il indique simplement "Erreur de syntaxe dans l'instruction de création de la table" sans autre précision sur sa nature et sa localisation.

Il existe heureusement une façon de réaliser le modèle physique sans être contraint aux exigences rigoureuses de la syntaxe du langage SQL. Cette deuxième approche, mieux adaptée au novice, est traitée dans la section suivante.

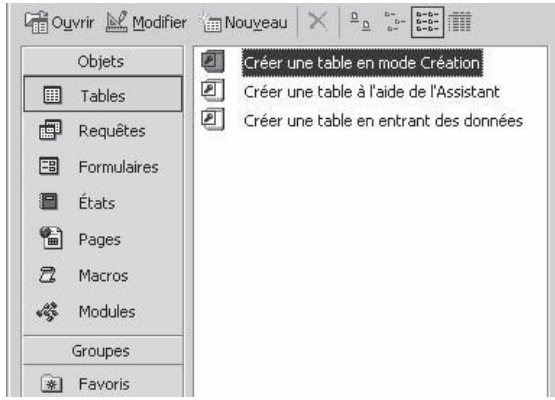
RÉALISATION LIMITÉE DU MODÈLE PHYSIQUE SANS FAIRE APPEL À SQL

Il est possible de réaliser un modèle physique en MS Access sans passer par un script SQL. Bien que moins souple qu'un script SQL, qui permet d'exprimer un spectre beaucoup plus large de contraintes du modèle logique, le mode dit *'Création de table'* de MS Access offre au novice un mécanisme simple et efficace pour réaliser et modifier un modèle physique de données.

Contrairement au script SQL, qui permet d'implanter un modèle physique par une séquence d'instructions exécutées dans une même procédure en une fraction de seconde, la réalisation d'un modèle physique sans faire appel à SQL doit se faire en deux temps :

- Premier temps : création de toutes les tables du modèle logique en mode *'Création de table'* ; chaque table doit avoir une clé primaire et le cas échéant une ou des clés étrangères ;
- Deuxième temps : établir les associations entre les tables en glissant la clé étrangère sur la clé primaire ou la clé secondaire à laquelle elle réfère à l'aide de l'outil ***Relations*** et spécifier le type d'intégrité référentielle applicable à chaque association.

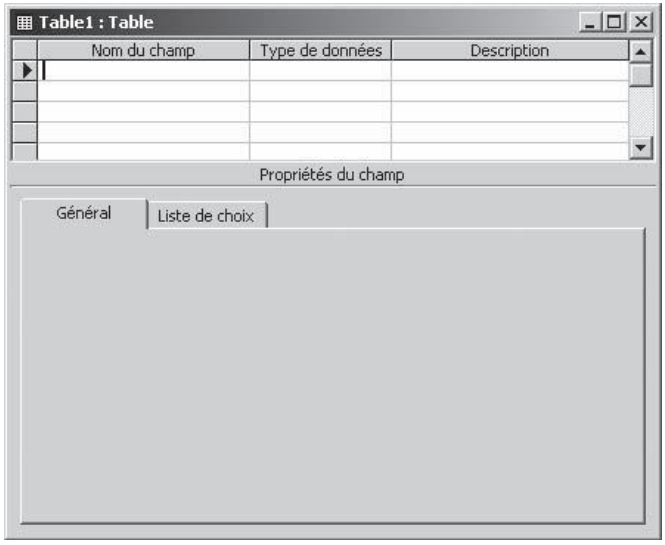
FIGURE 3-7 **Création d'une table en MS Access**



La création d'une table en MS Access se fait, comme pour tout autre type d'objet d'une BD Access, à travers la fenêtre de gestion des objets de la BD. La figure 3-7 reproduit la fenêtre qui, en MS Access, permet de gérer les objets avec ses catégories dans la marge à gauche.

La création d'une table s'effectue en sélectionnant le bouton *Tables* puis en double-cliquant sur *'Créer une table en mode Création'*, situé à droite dans la fenêtre. Une fenêtre comme celle illustrée à la figure 3-8 s'ouvre alors pour permettre au concepteur de créer une table en nommant chaque colonne puis en spécifiant leur type de données et les contraintes applicables à la colonne.

FIGURE 3-8 **Création d'une table en MS Access**



Temps 1 : Création de chaque table en mode 'Création de table'

Dans ce mode, chaque table doit être créée individuellement. Comme le montre la figure 3-8, MS Access utilise les termes *'Nom du champ'* pour faire référence au nom d'une colonne. Cette appellation est à notre avis inappropriée. Chaque colonne est définie par son nom et par un type de données choisi à travers une zone de liste déroulante.

On retrouve dans la liste déroulante les principaux types de données de la norme ANSI-92. Le tableau 3-9 nous indique sous quelle appellation chaque type ANSI est identifié dans la liste.

Lorsqu'un *champ* est sélectionné en marge à gauche de son nom, ses *propriétés* sont affichées au bas de la fenêtre. Les propriétés affichées sous l'onglet *Général* réfèrent notamment aux contraintes applicables à cette colonne.

TABLEAU 3-9 **Types de données en MS Access et correspondance en ANSI-92**

Type de données Access	Signification	Type ANSI-92
Numérique, taille du champ : ***Entier court***	Valeur numérique entière inférieure à 32 767	**SMALLINT**
Numérique, taille du champ : ***Entier long***	Valeur numérique entière inférieure à 2 147 483 647	**INTEGER**
Numérique, taille du champ : ***Réel simple***	Valeur numérique réelle inférieure à 3,402823E38	**FLOAT**
Numérique, taille du champ : ***Réel double***	Valeur numérique réelle inférieure à 1,79769313486231E308	**DOUBLE PRECISION**
Texte	Chaîne de 255 caractères ou moins	**VARCHAR**
Date/Heure	Date/heure du calendrier	**DATE**
Oui/Non	Valeur Vrai ou Faux	**BIT**
Monnaie	Valeur numérique réelle portant symbole monétaire	**CURRENCY**
Numéroauto	Valeur numérique entière sans doublons	**IDENTITY(d,i)**
Mémo	Chaîne de caractères de plus de 255 caractères, mais moins de 65 535	**MEMO**

La figure 3-9 montre une fenêtre contenant les spécifications de la table **Client** et, au bas de la fenêtre, les propriétés de la colonne **No client** sélectionnée plus haut. La présence de la petite clé en marge gauche du nom du champ **No client** (🔑), indique qu'il s'agit de la clé primaire de la table. Pour marquer un champ comme clé primaire, il faut au préalable sélectionner le champ et cliquer sur la même clé 🔑 dans la barre d'outils. S'il y a plusieurs de ces petites clés présentes pour une même table, l'ensemble des champs constitue *une clé primaire composée*. Une clé primaire composée est définie en sélectionnant au préalable simultanément les champs de la clé, EN TENANT LA TOUCHE CTRL ENFONCÉE AU COURS DE LA SÉLECTION MULTIPLE, et en cliquant enfin sur la même clé dans la barre d'outils.

Les propriétés du *champ* ou plus exactement d'une *colonne* reflètent les contraintes d'intégrité sémantiques données pour cette colonne au niveau logique par le modèle relationnel.

Le tableau 3-10 donne, pour chaque exigence que l'on peut retrouver dans le modèle relationnel, les modalités d'implantation dans le modèle physique à travers une ou des propriétés du champ. Mais toutes les exigences ne peuvent être mises en œuvre par le biais des propriétés des champs. En

FIGURE 3-9 **Création de la table *Client* en MS Access**

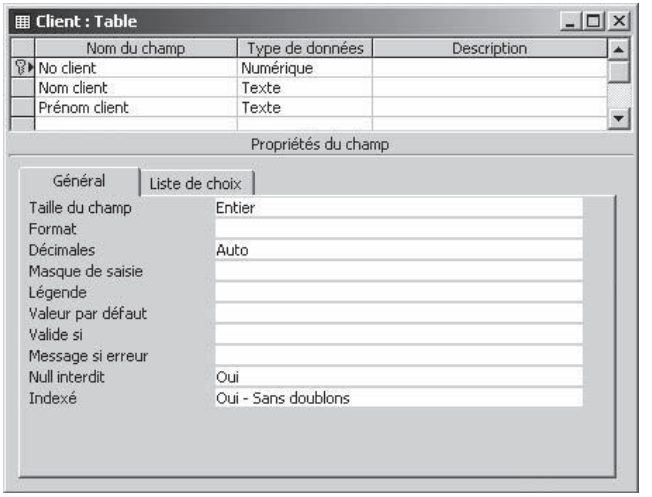


TABLEAU 3-10 **Exigences formulées au niveau logique et leur rendu en 'Mode création de table' dans les propriétés du champ**

Exigence	Niveau logique	Signification	Niveau physique : propriétés du champ utilisées
1.	{PK}	Cet attribut constitue la clé primaire : il ne peut avoir une valeur nulle et avoir des doublons (intégrité d'entité)	Présence en marge de ; Propriétés réglées ainsi : **Null interdit : *Oui*** **Indexé : *Oui- Sans doublons***
2.	{Non nul}	La valeur de cet attribut ne peut être nulle (laissée en blanc)	Propriété réglée ainsi : **Null interdit : *Oui***
3.	{FK}	Cet attribut constitue une clé étrangère avec intégrité référentielle en ajout	Doit être implanté au temps 2
4.	Multiplicité *minimale* 1 côté **mère**	La clé étrangère comporte obligatoirement une valeur	Propriété réglée ainsi : **Null interdit : *Oui*** pour la clé étrangère qui réalise l'association
7.	{>*Nombre*}	Cet attribut possède une contrainte de domaine	Propriété réglée ainsi : **Valide si : *>nombre***
8.	{Défaut *valeur*}	Cet attribut dispose d'une valeur par défaut	Propriété réglée ainsi : **Valeur par défaut : *valeur***
9.	Multiplicité *maximale* 1 côté **fille**	La clé étrangère ne peut avoir des doublons	Propriété réglée ainsi : **Indexé : *Oui- Sans doublons*** pour la clé étrangère qui réalise l'association

TABLEAU 3-10 **Exigences formulées au niveau logique et leur rendu en 'Mode création de table' dans les propriétés du champ (*suite*)**

Exigence	Niveau logique	Signification	Niveau physique : propriétés du champ utilisées
10.	enum{val_1, val_2, …, val_n }	Cet attribut possède une contrainte de domaine	Propriété réglée ainsi : **Valide si : *DANS(val_1, …, val_n)***
11.	{PK auto}	Cet attribut constitue une clé primaire avec génération automatique de valeurs	Le type de données du champ doit être **NuméroAuto** La génération de valeurs débute à 1 et suivent des valeurs incrémentées de 1
12.	{>*Nom_attribut$_2$*}	Cet attribut possède une contrainte voulant que sa valeur soit limitée par la valeur d'un attribut sur la même ligne	Cliquer sur le bouton *Propriétés* de la table de la barre d'outils  , une nouvelle fenêtre s'ouvre, la propriété suivante de la table est réglée ainsi : **Valise si ……** > *Nom_attribut$_2$*
13.	{<= *Valeur hors ligne*}	Cet attribut possède une contrainte voulant que sa valeur soit limitée par la valeur d'un attribut sur une autre ligne de la même table ou une ligne d'une autre table	Ne peut être réalisé en mode création de tables
14.	*Nom_attribut$_1$* {PK} … *Nom_attribut$_n$* {PK}	Ces attributs forment une clé primaire composée : aucun de ces attributs ne peut avoir une valeur nulle et leur combinaison ne peut avoir de doublons (intégrité d'entité)	Présence en marge de chaque champ de la clé  Propriétés réglées ainsi pour chaque champ de la clé composée : **Null interdit : *Oui***
15.	{FK}{Cascade}	Cet attribut constitue une clé étrangère avec intégrité référentielle en ajout, suppression et mise à jour	Doit être implanté au temps 2
16.	{*Nom_attribut$_1$: Type, …, Nom_attribut$_n$: Type*}{FK}	Ce groupe d'attributs constituent une clé étrangère avec intégrité référentielle en ajout	Doit être implanté au temps 2
17.	{Unique}	Attribut sans doublons	Propriété réglée ainsi : **Indexé : *Oui- Sans doublons***
18.	{PK, FK} L'attribut est le seul attribut de la clé primaire	Cet attribut est une clé primaire simple mais il est aussi clé étrangère.	Présence en marge de Propriétés réglées ainsi : **Null interdit : *Oui*** **Indexé : *Oui- Sans doublons***
19.	{ENTRE val_1 et val_2}		Propriété réglée ainsi : **Valide si : *ENTRE(val_1, …, val_n)***

effet, dans certains cas, il faudra attendre le temps 2 pour réaliser quelques-unes d'entre elles (exigences 3, 15 et 16). Dans d'autre cas, il sera impossible de le faire sans passer par un script SQL (exigence 13).

Les exigences du modèle relationnel qui ne peuvent être implantées dans un premier temps concernent essentiellement l'établissement du lien entre une clé étrangère et la clé primaire ou secondaire à laquelle la clé étrangère réfère. Elles seront mises en œuvre dès lors que toutes les tables du modèle relationnel auront été créées. Nous aborderons ceci à la prochaine section.

Temps 2 : Liaison des clés étrangères aux clés primaires des tables référencées

Avant d'aborder cette étape de la réalisation d'un modèle physique en MS Access sans script SQL, les tables du modèle relationnel doivent être toutes présentes dans la BD avec leur(s) clé(s) étrangère(s) s'il y a lieu.

C'est grâce à l'outil *Relations* de la barre d'outils que le travail de réalisation du modèle physique pourra être mené à son terme  . Une fenêtre *Relations* s'ouvre sur une fenêtre vide ou affiche les associations réalisées auparavant. Considérons le cas où aucune association n'existe encore. Comme le démontre la figure 3-10, la fenêtre *Relations* est initialement vide mais l'outil *Afficher la table* va permettre d'ajouter à la fenêtre, à tour de rôle, une représentation graphique de chaque table présente dans la BD.

FIGURE 3-10 **Ouverture de la fenêtre montrant les associations réalisées**

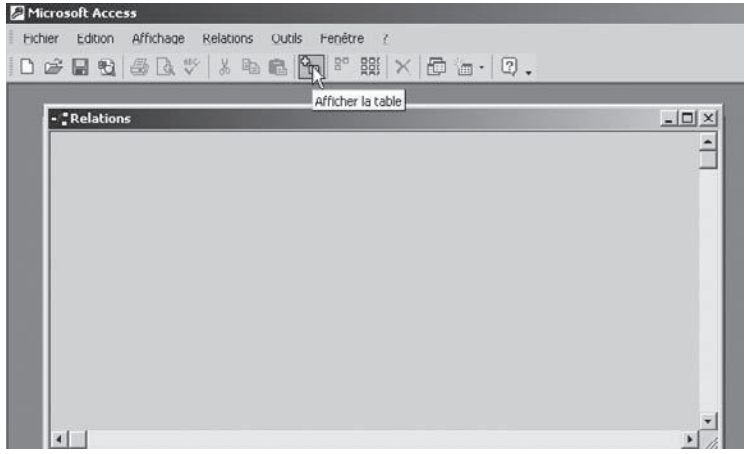
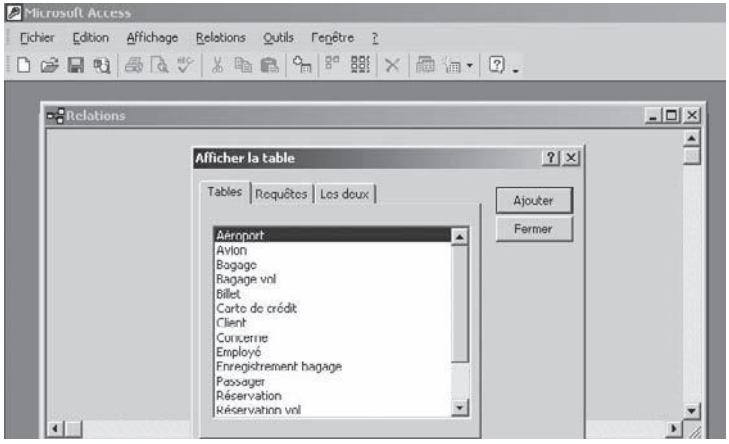


FIGURE 3-11 ***Afficher la table*** **permet d'ajouter à la fenêtre *Relations* les tables de la BD**



La figure 3-11 montre que MS Access fait apparaître par ordre alphabétique toutes les tables disponibles dans la BD. Il incombe à l'utilisateur d'ajouter à la fenêtre toutes les tables de la BD de manière à réaliser ensuite les associations.

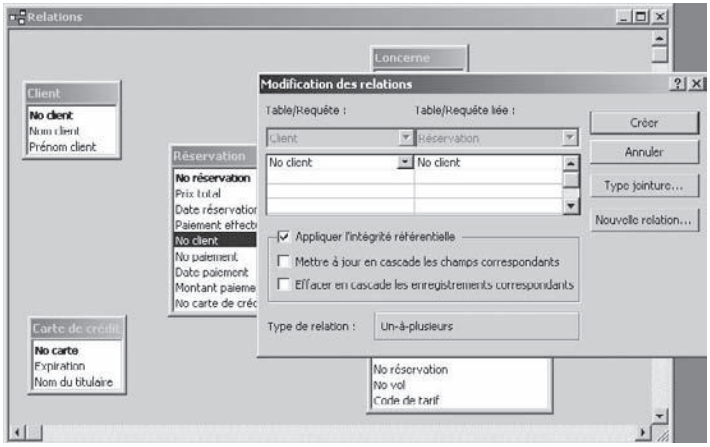
Lorsque les tables sont toutes affichées dans la fenêtre *Relations*, l'utilisateur peut implanter les associations en sélectionnant d'abord le nom d'une clé étrangère puis en la glissant sur le nom de la clé primaire ou secondaire de la table référencée.

Une boîte de dialogue s'ouvre alors, permettant à l'utilisateur d'établir le type d'intégrité référentielle à mettre en œuvre pour cette association. La figure 3-12 présente la boîte de dialogue *Modification des relations*, affichée dès lors que l'utilisateur a glissé le nom de la colonne **No client** de la table **Réservation** sur le nom de la colonne **No client** de la table **Client**. On note que l'utilisateur a le loisir de choisir la nature de l'intégrité référentielle à appliquer :

- en cochant *Appliquer l'intégrité référentielle*, elle sera appliquée en ajout
- en cochant *Mettre à jour en cascade*, elle sera appliquée en mise à jour
- en cochant *Effacer en cascade*, elle le sera en suppression.

L'utilisateur doit s'assurer que le type de données de la clé étrangère est le même que celui de la clé référencée dans l'autre table, sinon l'association va échouer. S'il s'agit d'une association impliquant la même table, soit une association réflexive, l'utilisateur doit afficher deux fois la table et procéder ensuite à l'association entre les deux occurrences de la même table. La deuxième fois qu'une table est affichée dans la fenêtre *Relations*, son nom est suivi de **_1** pour la distinguer de la première occurrence.

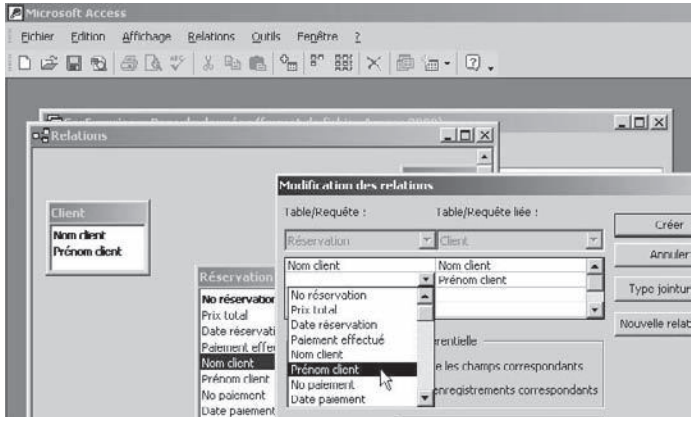
FIGURE 3-12 **Choix du type d'intégrité référentielle à appliquer sur l'association entre la table Client et la table Réservation**



Dans le cas particulier où la clé étrangère est une clé étrangère composée, l'utilisateur doit d'abord glisser un élément de la clé étrangère sur l'élément correspondant de la clé primaire référencée et ensuite, lorsque la boîte de dialogue *Modification des relations* s'affiche, les autres éléments de la clé sont choisis de part et d'autre de l'association à l'aide de zones de listes déroulantes.

La figure 3-13 présente une version du modèle de la figure 3-12 où la clé étrangère liant les tables **Réservation** à **Client** est cette fois composée de **Nom client** et **Prénom client**. L'utilisateur doit choisir dans les zones de liste déroulante les deux colonnes impliquées dans l'association de part et d'autre.

FIGURE 3-13 **Construction d'une association impliquant une clé étrangère composée**



VALIDATION DU MODÈLE PHYSIQUE DE DONNÉES

Le modèle physique réalisé à partir d'un modèle relationnel devrait être validé sur deux plans :

1. la cohérence avec le modèle relationnel ;
2. la validité du modèle sur le plan syntaxique et sémantique.

Le volet de validation le plus important est celui de la cohérence avec le modèle relationnel car seul le modélisateur peut l'effectuer. Ce volet concerne plus spécifiquement les points suivants :

- les tables du modèle relationnel doivent toutes se retrouver dans le modèle physique ;
- les colonnes d'une table du modèle relationnel doivent toutes être présentes dans la table correspondante du modèle physique ;
- la clé primaire d'une table du modèle relationnel doit être la même au niveau physique ;
- les clés étrangères reflètent toutes les associations d'une table fille et elles doivent se retrouver toutes dans la table correspondante au niveau physique avec leurs exigences d'intégrité référentielle ;
- chaque colonne d'une table du modèle physique possède un type de données et des contraintes d'intégrité conformes aux exigences du modèle relationnel.

La validation du modèle physique sur le plan syntaxique et sémantique est généralement prise en charge par les outils disponibles dans le SGBD pour implanter le modèle et ne relève donc pas directement du concepteur. Il s'agit notamment de détecter les anomalies suivantes :

- l'absence de clé primaire pour une table ;
- un type de données absent ou invalide ;
- une incompatibilité entre le type de données de la clé étrangère et le type de la clé référencée dans l'autre table ;
- dans le cas d'une clé étrangère composée, le nombre de colonnes et leur type respectif doivent être conformes à la clé composée référencée dans la deuxième table ;
- une formulation inadéquate des contraintes d'intégrité sémantique ;
- une clé primaire dont la colonne ou les colonnes ne portent pas la contrainte *Non nul* ;
- une clé primaire simple ou composée qui pourrait comporter des doublons.

Ces quelques considérations nous amènent à conclure le chapitre 3.

Le prochain chapitre se veut une démarche intégrée justifiant la pertinence et l'utilité des principes et des techniques de modélisation couverts au cours des trois premiers chapitres. L'objectif recherché est de présenter une méthode simplifiée, quoique complète et d'intérêt pratique, pour l'analyse, la conception et la réalisation d'une application de base de données. La méthode préconisée sera illustrée par de nombreux exemples. Elle reprend les principes et les techniques de modélisation pour élaborer l'ossature d'une application de bases de données cohérente et structurée.

EXERCICES DE MODÉLISATION PHYSIQUE DES DONNÉES

EXERCICE 3-1 Traduire le modèle relationnel de données (MRD) *optimisé,* présenté comme solution de l'exercice 2-1, en un modèle physique de données. Valider le modèle final et implanter le script en MS Access.

EXERCICE 3-2 Traduire le modèle relationnel de données (MRD) *non optimisé,* présenté comme solution de l'exercice 2-2, en un modèle physique de données. Ajouter à la table **Occupation fonction** une contrainte voulant que **Date entrée en fonction** soit toujours inférieure ou égale à **Date de départ de la fonction** et une autre contrainte voulant que **Salaire à la fin** soit strictement supérieur à zéro. Valider le modèle final et implanter le script en MS Access.

EXERCICE 3-3 Traduire le modèle relationnel de données (MRD) *optimisé*, présenté comme solution de l'exercice 2-3, en un modèle physique de données. Valider le modèle final et implanter le script en MS Access.

EXERCICE 3-4 Traduire le modèle relationnel de données (MRD) *optimisé*, présenté comme solution de l'exercice 2-4, en un modèle physique de données. Ajouter à la table **Processus** une contrainte voulant que le champ **Type** soit un *entier* entre 1 et 10. Valider le modèle final et implanter le script en MS Access.

EXERCICE 3-5 Traduire le modèle relationnel de données (MRD), présenté comme solution de l'exercice 2-5, en un modèle physique de données. Valider le modèle final et implanter le script en MS Access.

EXERCICE 3-6 Traduire le modèle relationnel de données (MRD) *optimisé*, présenté comme solution de l'exercice 2-6, en un modèle physique de données. Valider le modèle final et implanter le script en MS Access.

SOLUTIONS DES EXERCICES DE MODÉLISATION PHYSIQUE DES DONNÉES

EXERCICE 3-1

Étape 1.

Table	Exigence	Instruction SQL

Permis conduire
No permis : {PK}
Province
Pays
Téléphone client : {Non nul, Unique}
Nom client
Adresse client
	1
	
	2
	17
		`CREATE TABLE [Permis conduire]`
		`([No permis] SMALLINT`
		`NOT NULL PRIMARY KEY,`
		`[Province] VARCHAR(30),`
		`[Pays] VARCHAR(30),`
		`[Téléphone client] VARCHAR(12) NOT NULL`
		`UNIQUE,`
		`[Nom client] VARCHAR(30),`
		`[Adresse client] VARCHAR(50))`

Marque
Nom marque : {PK}
	1	`CREATE TABLE Marque`
		`([Nom marque] VARCHAR(30)`
		`NOT NULL PRIMARY KEY)`

Catégorie véhicule
Nom catégorie : {PK}
	1	`CREATE TABLE [Catégorie véhicule]`
		`([Nom catégorie] VARCHAR(30)`
		`NOT NULL PRIMARY KEY)`

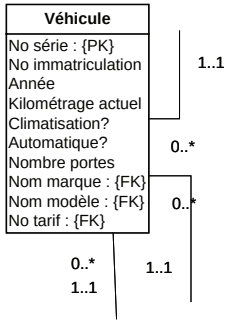
Tarif
No tarif : {PK}
Tarif horaire
Tarif quotidien
Tarif hebdomadaire
Tarif kilométrage
Tarif assurance
Montant franchise
Tarif suppression franchise
	1	`CREATE TABLE Tarif`
		`([No tarif] SMALLINT`
		`NOT NULL PRIMARY KEY,`
		`[Tarif horaire] CURRENCY,`
		`[Tarif quotidien] CURRENCY,`
		`[Tarif hebdomadaire] CURRENCY,`
		`[Tarif kilométrage] CURRENCY,`
		`[Tarif assurance] CURRENCY,`
		`[Montant franchise] CURRENCY,`
		`[Tarif suppression franchise] CURRENCY)`

Étape 2.

Modèle doit être créé avant Véhicule.

Table	Exigence	Instruction SQL

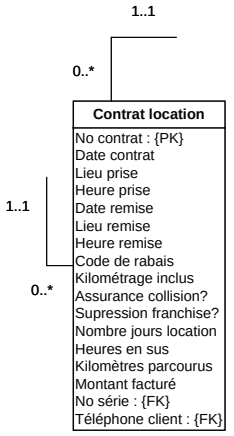
1..1
1..*
Modèle
Nom modèle : {PK}
Nom marque : {FK}
Nom catégorie : {FK}
1..*
1..1
	1
	
	4
	3
	
	4
	3
		`CREATE TABLE Modèle`
		`([Nom modèle] VARCHAR(30)`
		`NOT NULL PRIMARY KEY,`
		`[Nom marque] VARCHAR(30)`
		`NOT NULL`
		`REFERENCES Marque([Nom marque]),`
		`[Nom catégorie] VARCHAR(30)`
		`NOT NULL`
		`REFERENCES [Catégorie véhicule]([Nom catégorie]))`




```
                                CREATE TABLE Véhicule
                                 ([No série] SMALLINT
1                                NOT NULL PRIMARY KEY,
                                 [No immatriculation] SMALLINT,
                                 [Année] SMALLINT,
                                 [Kilométrage actuel] SMALLINT,
                                 [Climatisation?] BIT,
                                 [Automatique?] BIT,
                                 [Nombre portes] SMALLINT,
                                 [Nom marque] VARCHAR(30)
4                                NOT NULL
3                                REFERENCES Marque([Nom marque]),
                                 [Nom modèle] VARCHAR(30)
4                                NOT NULL
3                                REFERENCES Modèle([Nom modèle]),
                                 [No tarif] SMALLINT
4                                NOT NULL
3                                REFERENCES Tarif([No tarif]))
```

Étape 3.

Table	Exigence	Instruction SQL




```
                                CREATE TABLE [Contrant location]
                                 ([No contrat] SMALLINT
1                                NOT NULL PRIMARY KEY,
                                 [Date contrat] DATE,
                                 [Lieu prise] VARCHAR(30),
                                 [Heure prise] DATE,
                                 [Date remise] DATE,
                                 [Lieu remise] VARCHAR(30),
                                 [Heure remise] DATE,
                                 [Code rabais] VARCHAR(10),
                                 [Kilométrage inclus] SMALLINT,
                                 [Assurance collision?] BIT,
                                 [Suppression franchise?] BIT,
                                 [Nombre jours location] SMALLINT,
                                 [Heures en sus] VARCHAR(30),
                                 [Kilométrage parcourus] SMALLINT,
                                 [Montant facturé] CURRENCY,
                                 [No série] SMALLINT
4                                NOT NULL
3                                REFERENCES Véhicule([No série]),
                                 [Téléphone client] VARCHAR(12)
4                                NOT NULL
3                                REFERENCES [Permis conduire]([Téléphone
client]))
```

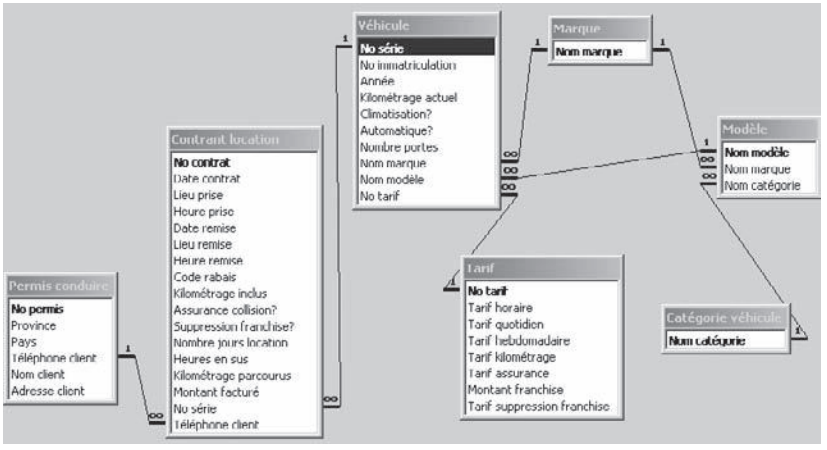
Script de réalisation du modèle en VBA.

```
Private Sub Script_Creation_Ex3_1()
Dim conDatabase As ADODB.Connection
Dim SQL As String

On Error GoTo Error_Handler
Set conDatabase = Application.CurrentProject.Connection

SQL = _
"CREATE TABLE [Permis conduire]" & _
" ([No permis] SMALLINT" & _
" NOT NULL PRIMARY KEY, " & _
" [Province] VARCHAR(30), " & _
" [Pays] VARCHAR(30), " & _
" [Téléphone client] VARCHAR(12) NOT NULL" & _
" UNIQUE, " & _
" [Nom client] VARCHAR(30), " & _
" [Adresse client] VARCHAR(50))"

conDatabase.Execute SQL
MsgBox "Table Permis conduire créée", vbInformation

SQL = _
"CREATE TABLE Marque" & _
" ([Nom marque] VARCHAR(30)" & _
" NOT NULL PRIMARY KEY)"

conDatabase.Execute SQL
MsgBox "Table Marque créée", vbInformation

SQL = _
"CREATE TABLE [Catégorie véhicule]" & _
" ([Nom catégorie] VARCHAR(30)" & _
" NOT NULL PRIMARY KEY)"

conDatabase.Execute SQL
MsgBox "Table Catégorie véhicule créée", vbInformation

SQL = _
"CREATE TABLE Tarif" & _
" ([No tarif] SMALLINT" & _
" NOT NULL PRIMARY KEY," & _
" [Tarif horaire] CURRENCY," & _
" [Tarif quotidien] CURRENCY," & _
" [Tarif hebdomadaire] CURRENCY," & _
" [Tarif kilométrage] CURRENCY," & _
" [Tarif assurance] CURRENCY," & _
" [Montant franchise] CURRENCY," & _
" [Tarif suppression franchise] CURRENCY)"
```

```
conDatabase.Execute SQL
MsgBox "Table Tarif créée", vbInformation

SQL = _
"CREATE TABLE Modèle" & _
" ([Nom modèle] VARCHAR(30) " & _
" NOT NULL PRIMARY KEY, " & _
" [Nom marque] VARCHAR(30) " & _
" NOT NULL" & _
" REFERENCES Marque([Nom marque]), " & _
" [Nom catégorie] VARCHAR(30)" & _
" NOT NULL" & _
" REFERENCES [Catégorie véhicule]([Nom catégorie]))"

conDatabase.Execute SQL
MsgBox "Table Modèle créée", vbInformation

SQL = _
"CREATE TABLE Véhicule" & _
" ([No série] SMALLINT" & _
" NOT NULL PRIMARY KEY, " & _
" [No immatriculation] SMALLINT," & _
" [Année] SMALLINT, " & _
" [Kilométrage actuel] SMALLINT, " & _
" [Climatisation?] BIT, " & _
" [Automatique?] BIT, " & _
" [Nombre portes] SMALLINT, " & _
" [Nom marque] VARCHAR(30) " & _
" NOT NULL" & _
" REFERENCES Marque([Nom marque]), " & _
" [Nom modèle] VARCHAR(30) " & _
" NOT NULL" & _
" REFERENCES Modèle([Nom modèle]), " & _
" [No tarif] SMALLINT " & _
" NOT NULL" & _
" REFERENCES Tarif([No tarif])) "

conDatabase.Execute SQL
MsgBox "Table Véhicule créée", vbInformation

SQL = _
"CREATE TABLE [Contrant location] " & _
" ([No contrat] SMALLINT" & _
" NOT NULL PRIMARY KEY, " & _
" [Date contrat] DATE, " & _
" [Lieu prise] VARCHAR(30), " & _
" [Heure prise] DATE, " & _
" [Date remise] DATE, " & _
" [Lieu remise] VARCHAR(30), " & _
" [Heure remise] DATE, " & _
```

```
" [Code rabais] VARCHAR(10), " & _
" [Kilométrage inclus] SMALLINT, " & _
" [Assurance collision?] BIT, " & _
" [Suppression franchise?] BIT, " & _
" [Nombre jours location] SMALLINT, " & _
" [Heures en sus] VARCHAR(30), " & _
" [Kilométrage parcourus] SMALLINT, " & _
" [Montant facturé] CURRENCY, " & _
" [No série] SMALLINT" & _
" NOT NULL" & _
" REFERENCES Véhicule([No série]), " & _
" [Téléphone client] VARCHAR(12) " & _
" NOT NULL" & _
" REFERENCES [Permis conduire]([Téléphone client]))"

conDatabase.Execute SQL
MsgBox "Table Contrat location créée", vbInformation

conDatabase.Close
Set conDatabase = Nothing

Exit Sub
Error_Handler:
 MsgBox Err.Description, vbInformation
End Sub
```

Schéma des associations dans MS Access.



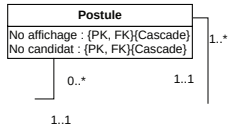
EXERCICE 3-2

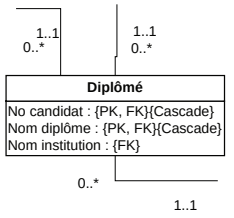
Étape 1.

Table	Exigence	Instruction SQL
Candidat No candidat : {PK} Nom candidat Prénom candidat No téléphone Date de naissance	1	`CREATE TABLE Candidat ([No candidat] SMALLINT NOT NULL PRIMARY KEY, [Nom candidat] VARCHAR(30), [Prénom candidat] VARCHAR(30), [No téléphone] VARCHAR(12), [Date de naissance] DATE)`
Diplôme Nom diplôme : {PK}	1	`CREATE TABLE Diplôme ([Nom diplôme] VARCHAR(30) NOT NULL PRIMARY KEY)`
Institution enseignement Nom institution : {PK}	1	`CREATE TABLE [Institution enseignement] ([Nom institution] VARCHAR(30) NOT NULL PRIMARY KEY)`
Langue Nom langue : {PK}	1	`CREATE TABLE Langue ([Nom langue] VARCHAR(30) NOT NULL PRIMARY KEY)`
Fonction Désignation : {PK}	1	`CREATE TABLE Fonction ([Désignation] VARCHAR(30) NOT NULL PRIMARY KEY)`
Poste No affichage : {PK} Nom poste	1	`CREATE TABLE Poste ([No affichage] SMALLINT NOT NULL PRIMARY KEY, [Nom poste] VARCHAR(30))`
Employeur Raison sociale : {PK}	1	`CREATE TABLE Employeur ([Raison sociale] VARCHAR(30) NOT NULL PRIMARY KEY)`
Centre d'intérêt Désignation : {PK}	1	`CREATE TABLE [Centre d'intérêt] ([Désignation] VARCHAR(30) NOT NULL PRIMARY KEY)`

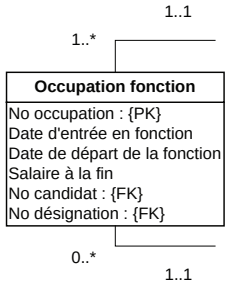
Étape 2.
N/A

Étape 3.

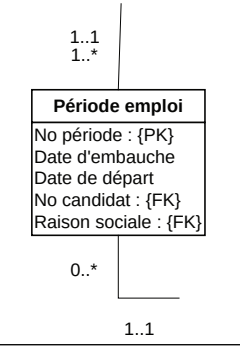
Table	Exigence	Instruction SQL
	14 15 14 15 14	`CREATE TABLE Postule ([No affichage] SMALLINT NOT NULL REFERENCES Poste([No affichage]) ON DELETE CASCADE ON UPDATE CASCADE, [No candidat] SMALLINT NOT NULL REFERENCES Candidat([No candidat]) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY([No affichage],[No candidat]))`



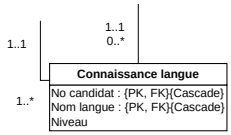
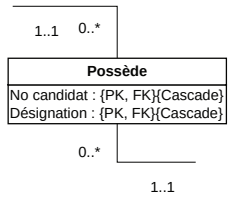

```
CREATE TABLE Diplômé
 ([No candidat] SMALLINT NOT NULL
14
15
 REFERENCES Candidat([No candidat])
 ON DELETE CASCADE
 ON UPDATE CASCADE,
14
 [Nom diplôme] VARCHAR(30) NULL
15
 REFERENCES Diplôme([Nom diplome])
 ON DELETE CASCADE
 ON UPDATE CASCADE,
 [Nom institution] VARCHAR(30)
 NOT NULL
4
 REFERENCES [Institution
3
enseignement]([Nom institution]),
 PRIMARY KEY([No candidat],[Nom
14
diplôme]))
```




```
CREATE TABLE [Occupation fonction]
 ([No occupation] SMALLINT
1
 NOT NULL PRIMARY KEY,
 [Date entrée en fonction] DATE,
 [Date de départ de la fonction] DATE,
 [Salaire à la fin] CURRENCY,
 [No candidat] SMALLINT
4
 NOT NULL
3
 REFERENCES Candidat([No candidat]),
 [Désignation] VARCHAR(30)
4
 NOT NULL
3
 REFERENCES Fonction([Désignation]),
12
 CONSTRAINT EntréeInfDépart CHECK([Date
entrée en fonction]<= [Date de départ de
la fonction]),
7
 CONSTRAINT SalSupZéro CHECK([Salaire à
la fin]>0))
```




```
CREATE TABLE [Période emploi]
 ([No période] SMALLINT
1
 NOT NULL PRIMARY KEY,
 [Date d'embauche] DATE,
 [Date de départ] DATE,
 [No candidat] SMALLINT
4
 NOT NULL
3
 REFERENCES Candidat([No candidat]),
 [Raison sociale] VARCHAR(30)
4
 NOT NULL
3
 REFERENCES Employeur([Raison sociale]))
```

	14 15 14 15 15	CREATE TABLE [Connaissance langue] ([No candidat] SMALLINT NOT NULL REFERENCES Candidat([No candidat]) ON DELETE CASCADE ON UPDATE CASCADE, [Nom langue] VARCHAR(30) NOT NULL REFERENCES Langue([Nom langue]) ON DELETE CASCADE ON UPDATE CASCADE, [Niveau] SMALLINT, PRIMARY KEY([No candidat],[Nom langue]))
	14 15 14 15 14	CREATE TABLE Possède ([No candidat] SMALLINT NOT NULL REFERENCES Candidat([No candidat]) ON DELETE CASCADE ON UPDATE CASCADE, [Désignation] VARCHAR(30) NOT NULL REFERENCES [Centre d'intérêt]([Désignation]) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY([No candidat],[Désignation]))

Script de réalisation du modèle en VBA.

```
Private Sub Script_Creation_Ex3_2()
Dim conDatabase As ADODB.Connection
Dim SQL As String

On Error GoTo Error_Handler
Set conDatabase = Application.CurrentProject.Connection

SQL = _
"CREATE TABLE Candidat" & _
" ([No candidat] SMALLINT" & _
" NOT NULL PRIMARY KEY," & _
" [Nom candidat] VARCHAR(30)," & _
" [Prénom candidat] VARCHAR(30)," & _
" [No téléphone] VARCHAR(12)," & _
" [Date de naissance] DATE)"

conDatabase.Execute SQL
MsgBox "Table Candidat créée", vbInformation

SQL = _
"CREATE TABLE Diplôme" & _
" ([Nom diplôme] VARCHAR(30)" & _
" NOT NULL PRIMARY KEY)"
```

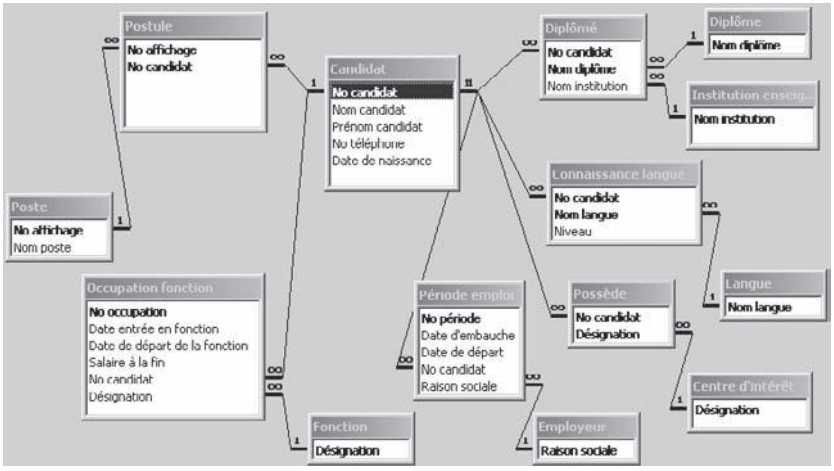
```
conDatabase.Execute SQL
MsgBox "Table Diplôme créée", vbInformation

SQL = _
"CREATE TABLE [Institution enseignement]" & _
" ([Nom institution] VARCHAR(30)" & _
" NOT NULL PRIMARY KEY)"

conDatabase.Execute SQL
MsgBox "Table Institution créée", vbInformation

SQL = _
"CREATE TABLE Langue" & _
" ([Nom langue] VARCHAR(30)" & _
" NOT NULL PRIMARY KEY)"

conDatabase.Execute SQL
MsgBox "Table Langue créée", vbInformation

SQL = _
"CREATE TABLE Fonction" & _
" ([Désignation] VARCHAR(30)" & _
" NOT NULL PRIMARY KEY)"

conDatabase.Execute SQL
MsgBox "Table Fonction créée", vbInformation

SQL = _
"CREATE TABLE Poste" & _
" ([No affichage] SMALLINT" & _
" NOT NULL PRIMARY KEY," & _
" [Nom poste] VARCHAR(30))"

conDatabase.Execute SQL
MsgBox "Table Poste créée", vbInformation

SQL = _
"CREATE TABLE Employeur" & _
" ([Raison sociale] VARCHAR(30)" & _
" NOT NULL PRIMARY KEY)"

conDatabase.Execute SQL
MsgBox "Table Employeur créée", vbInformation

SQL = _
"CREATE TABLE [Centre d'intérêt]" & _
" ([Désignation] VARCHAR(30)" & _
" NOT NULL PRIMARY KEY)"

conDatabase.Execute SQL
```

```
MsgBox "Table Centre intérêt créée", vbInformation

SQL = _
"CREATE TABLE Postule" & _
" ([No affichage] SMALLINT NOT NULL" & _
" REFERENCES Poste([No affichage])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [No candidat] SMALLINT NOT NULL" & _
" REFERENCES Candidat([No candidat])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" PRIMARY KEY([No affichage],[No candidat]))"

conDatabase.Execute SQL
MsgBox "Table Postule créée", vbInformation

SQL = _
"CREATE TABLE Diplômé" & _
" ([No candidat] SMALLINT NOT NULL" & _
" REFERENCES Candidat([No candidat])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [Nom diplôme] VARCHAR(30) NULL" & _
" REFERENCES Diplôme([Nom diplôme])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [Nom institution] VARCHAR(30)" & _
" NOT NULL" & _
" REFERENCES [Institution enseignement]([Nom institution])," & _
" PRIMARY KEY([No candidat],[Nom diplôme]))"

conDatabase.Execute SQL
MsgBox "Table Diplômé créée", vbInformation

SQL = _
"CREATE TABLE [Occupation fonction]" & _
" ([No occupation] SMALLINT " & _
" NOT NULL PRIMARY KEY," & _
" [Date entrée en fonction] DATE," & _
" [Date de départ de la fonction] DATE," & _
" [Salaire à la fin] CURRENCY," & _
" [No candidat] SMALLINT" & _
" NOT NULL" & _
" REFERENCES Candidat([No candidat])," & _
" [Désignation] VARCHAR(30)" & _
" NOT NULL" & _
" REFERENCES Fonction([Désignation])," & _
" CONSTRAINT EntréeInfDépart CHECK([Date entrée en fonction]<= [Date de
départ de la fonction])," & _
```

```
" CONSTRAINT SalSupZéro CHECK([Salaire à la fin]>0))"

conDatabase.Execute SQL
MsgBox "Table Occupation fonction créée", vbInformation

SQL = _
"CREATE TABLE [Période emploi]" & _
" ([No période] SMALLINT " & _
" NOT NULL PRIMARY KEY," & _
" [Date d'embauche] DATE," & _
" [Date de départ] DATE," & _
" [No candidat] SMALLINT" & _
" NOT NULL" & _
" REFERENCES Candidat([No candidat])," & _
" [Raison sociale] VARCHAR(30)" & _
" NOT NULL" & _
" REFERENCES Employeur([Raison sociale]))"

conDatabase.Execute SQL
MsgBox "Table Période emploi créée", vbInformation

SQL = _
"CREATE TABLE [Connaissance langue]" & _
" ([No candidat] SMALLINT NOT NULL" & _
" REFERENCES Candidat([No candidat])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [Nom langue] VARCHAR(30) NOT NULL" & _
" REFERENCES Langue([Nom langue])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [Niveau] SMALLINT," & _
" PRIMARY KEY([No candidat],[Nom langue]))"

conDatabase.Execute SQL
MsgBox "Table Connaissance langue créée", vbInformation

SQL = _
"CREATE TABLE Possède" & _
" ([No candidat] SMALLINT NOT NULL" & _
" REFERENCES Candidat([No candidat])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [Désignation] VARCHAR(30) NOT NULL" & _
" REFERENCES [Centre d'intérêt]([Désignation])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" PRIMARY KEY([No candidat],[Désignation]))"

conDatabase.Execute SQL
```

```
MsgBox "Table Possèdecréée", vbInformation

conDatabase.Close
Set conDatabase = Nothing
Exit Sub
Error_Handler:
 MsgBox Err.Description, vbInformation
End Sub
```

Schéma des associations dans MS Access.

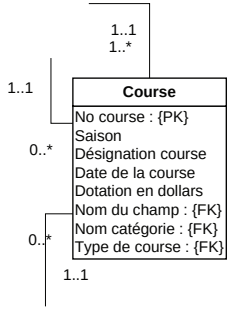


EXERCICE 3-3

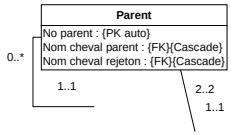
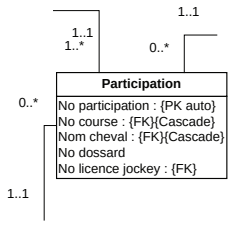
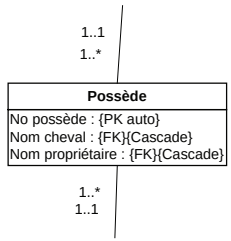
Étape 1.

Table	Exigence	Instruction SQL
Champ de course Nom du champ : {PK}	1	`CREATE TABLE [Champ de course] ([Nom du champ] VARCHAR(30) NOT NULL PRIMARY KEY)`
Catégorie course Nom catégorie : {PK}	1	`CREATE TABLE [Catégorie course] ([Nom catégorie] VARCHAR(30) NOT NULL PRIMARY KEY)`
Type de course Nom type : {PK}	1	`CREATE TABLE [Type de course] ([Nom type] VARCHAR(30) NOT NULL PRIMARY KEY)`
Cheval Nom cheval : {PK} Sexe cheval Date naissance cheval Gains cette saison	1	`CREATE TABLE Cheval ([Nom cheval] VARCHAR(30) NOT NULL PRIMARY KEY, [Sexe cheval] VARCHAR(1), [Date naissance cheval] DATE, [Gains cette saison] CURRENCY)`
Jockey No licence jockey : {PK} Nom jockey	1	`CREATE TABLE Jockey ([No licence jockey] SMALLINT NOT NULL PRIMARY KEY, [Nom jockey] VARCHAR(30))`
Propiétaire Nom propriétaire : {PK}	1	`CREATE TABLE Propriétaire ([Nom propriétaire] VARCHAR(30) NOT NULL PRIMARY KEY)`

Étape 2.

Table	Exigence	Instruction SQL
	1 4 3 4 3 4 3	`CREATE TABLE Course ([No course] SMALLINT NOT NULL PRIMARY KEY, [Saison] VARCHAR(30), [Désignation cours] VARCHAR(30), [Date de la course] DATE, [Dotation en dollars] CURRENCY, [Nom du champ] VARCHAR(30) NOT NULL REFERENCES [Champ de course]([Nom du champ]), [Nom catégorie] VARCHAR(30) NOT NULL REFERENCES [Catégorie course]([Nom catégorie]) [Nom type] VARCHAR(30) NOT NULL REFERENCES [Type de course]([Nom type]))`

Étape 3.
Note : La multiplicité 2..2 est en fait du type 1..*.

Table	Exigence	Instruction SQL
	 11 14 15 14 15	CREATE TABLE Parent ([No parent] IDENTITY NOT NULL PRIMARY KEY, [Nom cheval parent] VARCHAR(30) NOT NULL REFERENCES Cheval([Nom cheval]) ON DELETE CASCADE ON UPDATE CASCADE, [Nom cheval rejeton] VARCHAR(30) NOT NULL REFERENCES Cheval([Nom cheval]) ON DELETE CASCADE ON UPDATE CASCADE)
	 11 14 15 14 15 4 3	CREATE TABLE Participation ([No participation] IDENTITY NOT NULL PRIMARY KEY, [No course] SMALLINT NOT NULL REFERENCES Course([No course]) ON DELETE CASCADE ON UPDATE CASCADE, [Nom cheval] VARCHAR(30) NOT NULL REFERENCES Cheval([Nom cheval]) ON DELETE CASCADE ON UPDATE CASCADE, [No dossard] SMALLINT, [No licence jockey] SMALLINT NOT NULL REFERENCES Jockey([No licence jockey]))
	 11 14 15 14 15	CREATE TABLE Possède ([No possède] IDENTITY NOT NULL PRIMARY KEY, [Nom cheval] VARCHAR(30) NOT NULL REFERENCES Cheval([Nom cheval]) ON DELETE CASCADE ON UPDATE CASCADE, [Nom propriétaire] VARCHAR(30) NOT NULL REFERENCES Propriétaire([Nom propriétaire]) ON DELETE CASCADE ON UPDATE CASCADE)

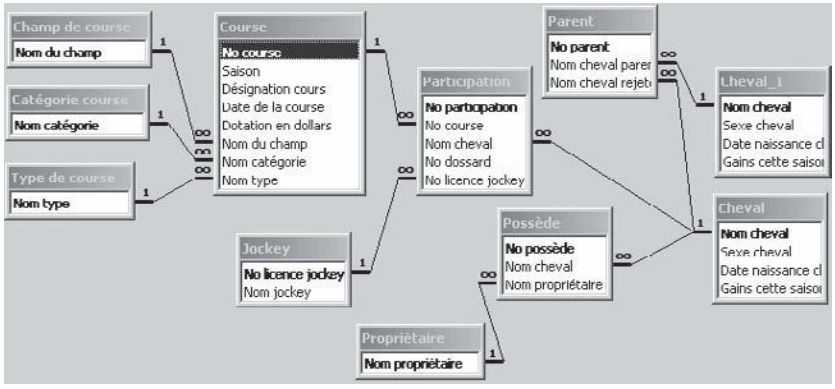
Script de réalisation du modèle en VBA.

```
Private Sub Script_Creation_Ex3_3()
Dim conDatabase As ADODB.Connection
Dim SQL As String

On Error GoTo Error_Handler
Set conDatabase = Application.CurrentProject.Connection

SQL = _
"CREATE TABLE [Champ de course]" & _
" ([Nom du champ] VARCHAR(30)" & _
" NOT NULL PRIMARY KEY)"

conDatabase.Execute SQL
MsgBox "Table Champ de course créée", vbInformation

SQL = _
"CREATE TABLE [Catégorie course]" & _
" ([Nom catégorie] VARCHAR(30)" & _
" NOT NULL PRIMARY KEY)"
conDatabase.Execute SQL
MsgBox "Table Catégorie créée", vbInformation

SQL = _
"CREATE TABLE [Type de course]" & _
" ([Nom type] VARCHAR(30)" & _
" NOT NULL PRIMARY KEY)"

conDatabase.Execute SQL
MsgBox "Table Type de course créée", vbInformation

SQL = _
"CREATE TABLE Cheval" & _
" ([Nom cheval] VARCHAR(30)" & _
" NOT NULL PRIMARY KEY," & _
" [Sexe cheval] VARCHAR(1)," & _
" [Date naissance cheval] DATE," & _
" [Gains cette saison] CURRENCY)"

conDatabase.Execute SQL
MsgBox "Table Cheval créée", vbInformation

SQL = _
"CREATE TABLE Jockey" & _
" ([No licence jockey] SMALLINT" & _
" NOT NULL PRIMARY KEY," & _
" [Nom jockey] VARCHAR(30))"

conDatabase.Execute SQL
MsgBox "Table Jockey créée", vbInformation
SQL = _
"CREATE TABLE Propriétaire" & _
```

```
" ([Nom propriétaire] VARCHAR(30)" & _
" NOT NULL PRIMARY KEY)"

conDatabase.Execute SQL
MsgBox "Table Propriétaire créée", vbInformation

SQL = _
"CREATE TABLE Course" & _
" ([No course] SMALLINT" & _
" NOT NULL PRIMARY KEY," & _
" [Saison] VARCHAR(30)," & _
" [Désignation cours] VARCHAR(30)," & _
" [Date de la course] DATE," & _
" [Dotation en dollars] CURRENCY," & _
" [Nom du champ] VARCHAR(30) NOT NULL" & _
" REFERENCES [Champ de course]([Nom du champ])," & _
" [Nom catégorie] VARCHAR(30) NOT NULL" & _
" REFERENCES [Catégorie course]([Nom catégorie])," & _
" [Nom type] VARCHAR(30) NOT NULL" & _
" REFERENCES [Type de course]([Nom type]))"

conDatabase.Execute SQL
MsgBox "Table Course créée", vbInformation

SQL = _
"CREATE TABLE Parent" & _
" ([No parent] IDENTITY" & _
" NOT NULL PRIMARY KEY," & _
" [Nom cheval parent] VARCHAR(30) " & _
" NOT NULL" & _
" REFERENCES Cheval([Nom cheval])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [Nom cheval rejeton] VARCHAR(30) " & _
" NOT NULL" & _
" REFERENCES Cheval([Nom cheval])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE)"

conDatabase.Execute SQL
MsgBox "Table Parent créée", vbInformation

SQL = _
"CREATE TABLE Participation" & _
" ([No participation] IDENTITY" & _
" NOT NULL PRIMARY KEY," & _
" [No course] SMALLINT " & _
" NOT NULL" & _
" REFERENCES Course([No course])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [Nom cheval] VARCHAR(30) " & _
" NOT NULL" & _
```

```
" REFERENCES Cheval([Nom cheval])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [No dossard] SMALLINT," & _
" [No licence jockey] SMALLINT" & _
" NOT NULL" & _
" REFERENCES Jockey([No licence jockey]))"

conDatabase.Execute SQL
MsgBox "Table Participation créée", vbInformation
SQL = _
"CREATE TABLE Possède" & _
" ([No possède] IDENTITY" & _
" NOT NULL PRIMARY KEY," & _
" [Nom cheval] VARCHAR(30) " & _
" NOT NULL" & _
" REFERENCES Cheval([Nom cheval])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [Nom propriétaire] VARCHAR(30) " & _
" NOT NULL" & _
" REFERENCES Propriétaire([Nom propriétaire])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE)"

conDatabase.Execute SQL
MsgBox "Table Possède créée", vbInformation

conDatabase.Close
Set conDatabase = Nothing
Exit Sub
Error_Handler:
 MsgBox Err.Description, vbInformation
End Sub
```

Schéma des associations dans MS Access.



EXERCICE 3-4

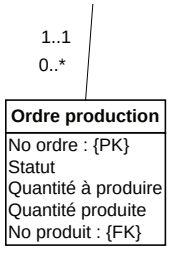
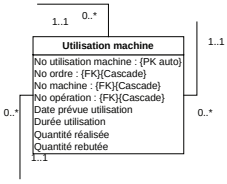
Étape 1.

Table	Exigence	Instruction SQL
Processus No processus : {PK} Type Responsable Date dernier changement	1 19	`CREATE TABLE Processus ([No processus] SMALLINT NOT NULL PRIMARY KEY, [Type] SMALLINT, [Responsable] VARCHAR(30), [Date dernier changement] DATE, CONSTRAINT TypeValide CHECK(Type BETWEEN 1 AND 10)`
Outillage No outillage : {PK} Description		`CREATE TABLE Outillage ([No Outillage] SMALLINT NOT NULL PRIMARY KEY, [Description] VARCHAR(30))`
Machine No machine : {PK} Description Coût utilisation/min.		`CREATE TABLE Machine ([No machine] SMALLINT NOT NULL PRIMARY KEY, [Description] VARCHAR(30), [Coût utilisation/min] CURRENCY)`

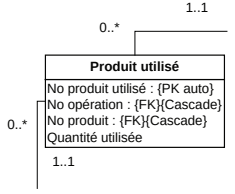
Étape 2.

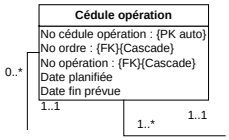
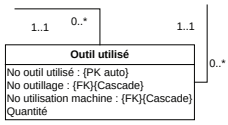
Mise en garde : toute clé étrangère liée à une clé primaire de type IDENTITY doit être de type INTEGER (entier long)

Table	Exigence	Instruction SQL
1..1 1..* **Opération** No opération : {PK auto} No processus : {FK}{Cascade} No séquence Description Temps standard	11 4 15	`CREATE TABLE Opération ([No opération] IDENTITY NOT NULL PRIMARY KEY, [No processus] SMALLINT NOT NULL REFERENCES Processus([No processus]) ON DELETE CASCADE ON UPDATE CASCADE, [No séquence] SMALLINT, [Description] VARCHAR(30), [Temps standard] FLOAT)`
0..1 1..* **Produit** No produit : {PK} No processus : {FK}	1 3	`CREATE TABLE Produit ([No produit] SMALLINT NOT NULL PRIMARY KEY, [No processus] SMALLINT REFERENCES Processus([No processus]))`

	 1 4 3	CREATE TABLE [Ordre production] ([No ordre] SMALLINT NOT NULL PRIMARY KEY, [Quantité à produire] SMALLINT, [Quantité produite] SMALLINT, [No produit] SMALLINT NOT NULL REFERENCES Produit([No produit]))
	11 4 15 4 15 5 15	CREATE TABLE [Utilisation machine] ([No utilisation machine] IDENTITY NOT NULL PRIMARY KEY, [No ordre] SMALLINT NOT NULL REFERENCES [Ordre production]([No ordre]) ON DELETE CASCADE ON UPDATE CASCADE, [No machine] SMALLINT NOT NULL REFERENCES Machine([No machine]) ON DELETE CASCADE ON UPDATE CASCADE, [No opération] INTEGER NOT NULL REFERENCES Opération([No opération]) ON DELETE CASCADE ON UPDATE CASCADE, [Date prévue utilisation] DATE, [Durée utilisation] FLOAT, [Quantité réalisée] SMALLINT, [Quantité rebutée] SMALLINT)

Étape 3.

Table	Exigence	Instruction SQL
	 11 14 15 14 15 	CREATE TABLE [Produit utilisé] ([No produit utilisé] IDENTITY NOT NULL PRIMARY KEY, [No opération] INTEGER NOT NULL REFERENCES Opération([No opération]) ON DELETE CASCADE ON UPDATE CASCADE, [No produit] SMALLINT NOT NULL REFERENCES Produit([No produit]) ON DELETE CASCADE ON UPDATE CASCADE, [Quantité utilisée] SMALLINT)

	11 14 15 14 15	CREATE TABLE [Cédule opération] ([No cédule opération] IDENTITY NOT NULL PRIMARY KEY, [No ordre] SMALLINT NOT NULL REFERENCES [Ordre production]([No ordre]) ON DELETE CASCADE ON UPDATE CASCADE, [No opération] INTEGER NOT NULL REFERENCES Opération([No opération]) ON DELETE CASCADE ON UPDATE CASCADE, [Date planifiée] DATE, [Date fin prévue] DATE)
	11 14 15 14 15	CREATE TABLE [Outil utilisé] ([No outil utilisé] IDENTITY NOT NULL PRIMARY KEY, [No outillage] SMALLINT NOT NULL REFERENCES Outillage([No outillage]) ON DELETE CASCADE ON UPDATE CASCADE, [No utilisation machine] INTEGER NOT NULL REFERENCES [Utilisation machine]([No utilisation machine]) ON DELETE CASCADE ON UPDATE CASCADE, [Quantité] SMALLINT)

Script de réalisation du modèle en VBA.

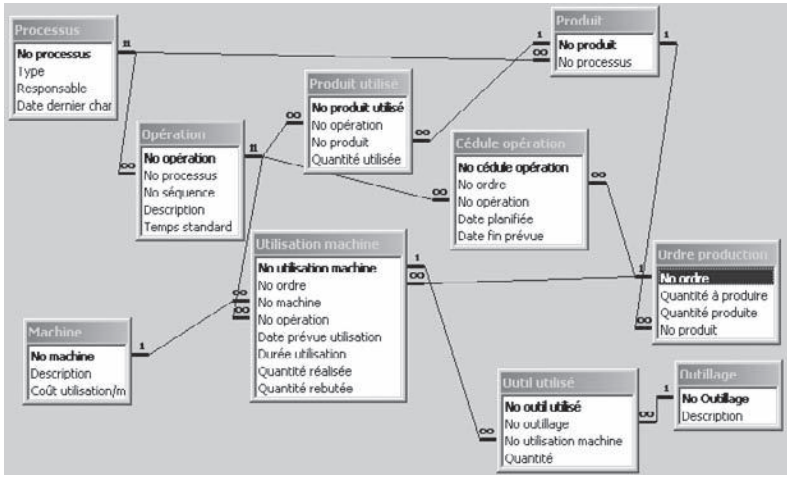
```
Private Sub Script_Creation_Ex3_4()
Dim conDatabase As ADODB.Connection
Dim SQL As String

On Error GoTo Error_Handler
Set conDatabase = Application.CurrentProject.Connection

SQL = _
"CREATE TABLE Processus" & _
" ([No processus] SMALLINT" & _
" NOT NULL PRIMARY KEY," & _
" [Type] SMALLINT," & _
" [Responsable] VARCHAR(30)," & _
" [Date dernier changement] DATE," & _
" CONSTRAINT TypeValide CHECK(Type BETWEEN 1 AND 10))"

conDatabase.Execute SQL
MsgBox "Table Processus créée", vbInformation
```

```
SQL = _
"CREATE TABLE Outillage" & _
" ([No Outillage] SMALLINT" & _
" NOT NULL PRIMARY KEY," & _
" [Description] VARCHAR(30))"

conDatabase.Execute SQL
MsgBox "Table Outillage créée", vbInformation

SQL = _
"CREATE TABLE Machine" & _
" ([No machine] SMALLINT" & _
" NOT NULL PRIMARY KEY," & _
" [Description] VARCHAR(30)," & _
" [Coût utilisation/min] CURRENCY)"

conDatabase.Execute SQL
MsgBox "Table Machine créée", vbInformation

SQL = _
"CREATE TABLE Opération" & _
" ([No opération] IDENTITY " & _
" NOT NULL PRIMARY KEY," & _
" [No processus] SMALLINT NOT NULL" & _
" REFERENCES Processus([No processus])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [No séquence] SMALLINT," & _
" [Description] VARCHAR(30)," & _
" [Temps standard] FLOAT)"

conDatabase.Execute SQL
MsgBox "Table Opération créée", vbInformation

SQL = _
"CREATE TABLE Produit" & _
" ([No produit] SMALLINT " & _
" NOT NULL PRIMARY KEY," & _
" [No processus] SMALLINT" & _
" REFERENCES Processus([No processus]))"

conDatabase.Execute SQL
MsgBox "Table Produit créée", vbInformation

SQL = _
"CREATE TABLE [Ordre production]" & _
" ([No ordre] SMALLINT " & _
" NOT NULL PRIMARY KEY," & _
" [Quantité à produire] SMALLINT," & _
" [Quantité produite] SMALLINT," & _
" [No produit] SMALLINT NOT NULL" & _
" REFERENCES Produit([No produit]))"
```

```
conDatabase.Execute SQL
MsgBox "Table Ordre production créée", vbInformation

SQL = _
"CREATE TABLE [Utilisation machine]" & _
" ([No utilisation machine] IDENTITY " & _
" NOT NULL PRIMARY KEY," & _
" [No ordre] SMALLINT NOT NULL" & _
" REFERENCES [Ordre production]([No ordre])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [No machine] SMALLINT NOT NULL" & _
" REFERENCES Machine([No machine])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [No opération] INTEGER NOT NULL" & _
" REFERENCES Opération([No opération])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [Date prévue utilisation] DATE," & _
" [Durée utilisation] FLOAT," & _
" [Quantité réalisée] SMALLINT," & _
" [Quantité rebutée] SMALLINT)"

conDatabase.Execute SQL
MsgBox "Table Utilisation machine créée", vbInformation

SQL = _
"CREATE TABLE [Produit utilisé]" & _
" ([No produit utilisé] IDENTITY " & _
" NOT NULL PRIMARY KEY," & _
" [No opération] INTEGER NOT NULL" & _
" REFERENCES Opération([No opération])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [No produit] SMALLINT NOT NULL" & _
" REFERENCES Produit([No produit])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [Quantité utilisée] SMALLINT)"

conDatabase.Execute SQL
MsgBox "Table Produit utilisé créée", vbInformation

SQL = _
"CREATE TABLE [Cédule opération]" & _
" ([No cédule opération] IDENTITY " & _
" NOT NULL PRIMARY KEY," & _
" [No ordre] SMALLINT NOT NULL" & _
" REFERENCES [Ordre production]([No ordre])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [No opération] INTEGER NOT NULL" & _
```

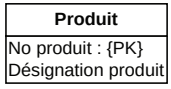
```
" REFERENCES Opération([No opération])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [Date planifiée] DATE," & _
" [Date fin prévue] DATE)"

conDatabase.Execute SQL
MsgBox "Table Cédule opération créée", vbInformation

SQL = _
"CREATE TABLE [Outil utilisé]" & _
" ([No outil utilisé] IDENTITY " & _
" NOT NULL PRIMARY KEY," & _
" [No outillage] SMALLINT NOT NULL" & _
" REFERENCES Outillage([No outillage])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [No utilisation machine] INTEGER NOT NULL" & _
" REFERENCES [Utilisation machine]([No utilisation machine])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [Quantité] SMALLINT)"

conDatabase.Execute SQL
MsgBox "Table Outil utilisé créée", vbInformation

conDatabase.Close
Set conDatabase = Nothing
Exit Sub
Error_Handler:
 MsgBox Err.Description, vbInformation
End Sub
```

Schéma des associations dans MS Access.

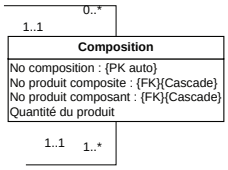


EXERCICE 3-5

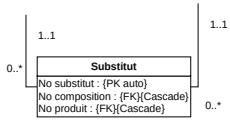
Étape 1.

Table	Exigence	Instruction SQL
	1	CREATE TABLE Produit ([No produit] SMALLINT NOT NULL PRIMARY KEY, [Désignation produit] VARCHAR(30))

Étape 2.

Table	Exigence	Instruction SQL
	1 4 3 4 3	CREATE TABLE Composition ([No composition] IDENTITY NOT NULL PRIMARY KEY, [No produit composite] SMALLINT NOT NULL REFERENCES Produit([No produit]), [No produit composant] SMALLINT NOT NULL REFERENCES Produit([No produit]))

Étape 3.

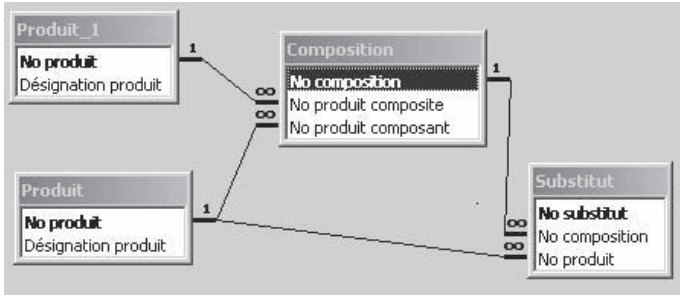
Table	Exigence	Instruction SQL
	11 14 15 14 15	CREATE TABLE Substitut ([No substitut] IDENTITY NOT NULL PRIMARY KEY, [No composition] INTEGER NOT NULL REFERENCES Composition([No composition]) ON DELETE CASCADE ON UPDATE CASCADE, [No produit] SMALLINT NOT NULL REFERENCES Produit([No produit]) ON DELETE CASCADE ON UPDATE CASCADE)

Script de réalisation du modèle en VBA.

```
Private Sub Script_Creation_Ex3_5()
Dim conDatabase As ADODB.Connection
Dim SQL As String

On Error GoTo Error_Handler
Set conDatabase = Application.CurrentProject.Connection

SQL = _
"CREATE TABLE Produit" & _
" ([No produit] SMALLINT" & _
" NOT NULL PRIMARY KEY," & _
" [Désignation produit] VARCHAR(30))"

conDatabase.Execute SQL
```

```
MsgBox "Table Produit créée", vbInformation

SQL = _
"CREATE TABLE Composition" & _
" ([No composition] IDENTITY" & _
" NOT NULL PRIMARY KEY," & _
" [No produit composite] SMALLINT" & _
" NOT NULL" & _
" REFERENCES Produit([No produit])," & _
" [No produit composant] SMALLINT" & _
" NOT NULL" & _
" REFERENCES Produit([No produit]))"

conDatabase.Execute SQL
MsgBox "Table Composition créée", vbInformation

SQL = _
"CREATE TABLE Substitut" & _
" ([No substitut] IDENTITY" & _
" NOT NULL PRIMARY KEY," & _
" [No composition] INTEGER NOT NULL" & _
" REFERENCES Composition([No composition])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [No produit] SMALLINT NOT NULL" & _
" REFERENCES Produit([No produit])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE)"

conDatabase.Execute SQL
MsgBox "Table Substitut créée", vbInformation


conDatabase.Close
Set conDatabase = Nothing
Exit Sub
Error_Handler:
 MsgBox Err.Description, vbInformation
End Sub
```

Schéma des associations dans MS Access.



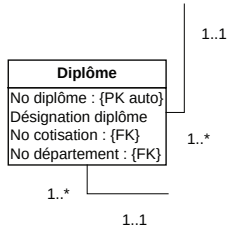
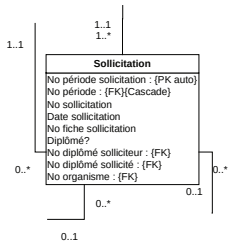
EXERCICE 3-6

Étape 1.

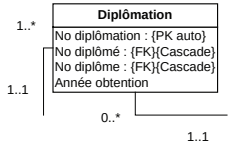
Table	Exigence	Instruction SQL
Diplômé No diplômé : {PK} Nom diplômé Prénom diplômé Date de naissance Téléphone bureau Téléphone domicile Adresse domicile Adresse travail Membre CA? Membre exécutif?	1	`CREATE TABLE Diplômé ([No diplômé] SMALLINT NOT NULL PRIMARY KEY, [Nom diplômé] VARCHAR(30), [Prénom diplômé] VARCHAR(30), [Téléphone bureau] VARCHAR(12), [Date de naissance] DATE, [Téléphone domicile] VARCHAR(12), [Adresse domicile] VARCHAR(50), [Adresse travail] VARCHAR(50), [Membre CA?] BIT, [Membre exécutif?] BIT)`
Type de cotisation No cotisation : {PK} Montant annuel	1	`CREATE TABLE [Type cotisation] ([No cotisation] SMALLINT NOT NULL PRIMARY KEY, [Montant annuel] CURRENCY)`
Faculté No faculté : {PK auto} Nom faculté	11	`CREATE TABLE Faculté ([No faculté] IDENTITY NOT NULL PRIMARY KEY, [Nom faculté] VARCHAR(30))`
Période de campagne No période : {PK} Année campagne Date début Date fin	1	`CREATE TABLE [Période de campagne] ([No période] SMALLINT NOT NULL PRIMARY KEY, [Année campagne] SMALLINT, [Date début] DATE, [Date fin] DATE)`
Organisme No organisme : {PK auto} Nom organisme Premier téléphone Deuxième téléphone	11	`CREATE TABLE Organisme ([No organisme] IDENTITY NOT NULL PRIMARY KEY, [Nom organisme] VARCHAR(30), [Téléphone premier] VARCHAR(12), [Téléphone deuxième] VARCHAR(12))`

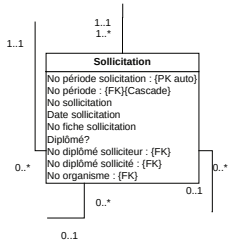
Étape 2.

Table	Exigence	Instruction SQL
Département No département : {PK auto} Nom département No faculté : {FK} 1..* 1..1	1 4 3	`CREATE TABLE Département ([No département] IDENTITY NOT NULL PRIMARY KEY, [Nom département] VARCHAR(30), [No faculté] INTEGER NOT NULL REFERENCES Faculté([No faculté]))`

	11 4 3 4 3	CREATE TABLE Diplôme ([No diplôme] IDENTITY NOT NULL PRIMARY KEY, [Désignation diplôme] VARCHAR(30), [No cotisation] SMALLINT NOT NULL REFERENCES [Type cotisation]([No cotisation]), [No département] INTEGER NOT NULL REFERENCES Département([No département]))
	11 4 15 3 4 3 3	CREATE TABLE Sollicitation ([No période sollicitation] IDENTITY NOT NULL PRIMARY KEY, [No période] SMALLINT NOT NULL REFERENCES [Période de campagne]([No période]) ON DELETE CASCADE ON UPDATE CASCADE, [No sollicitation] SMALLINT, [Date sollicitation] DATE, [No fiche sollicitation] SMALLINT, [Diplômé?] BIT, [No diplômé solliciteur] SMALLINT REFERENCES Diplômé([No diplômé]), [No diplômé sollicité] SMALLINT NOT NULL REFERENCES Diplômé([No diplômé]), [No organisme] INTEGER REFERENCES Organisme([No organisme]))

Étape 3.

Table	Exigence	Instruction SQL
	11 14 15 14 15	CREATE TABLE Diplômation ([No diplômation] IDENTITY NOT NULL PRIMARY KEY, [No diplômé] SMALLINT NOT NULL REFERENCES Diplômé([No diplômé]) ON DELETE CASCADE ON UPDATE CASCADE, [No diplôme] INTEGER NOT NULL REFERENCES Diplôme([No diplôme]) ON DELETE CASCADE ON UPDATE CASCADE, [Année obtention] DATE)

	1 3 3	CREATE TABLE Versement ([No versement] SMALLINT NOT NULL PRIMARY KEY, [Date versement] DATE, [Montant versement] CURRENCY, [Cotisation?] BIT, [Année] SMALLINT, [No diplômé] SMALLINT REFERENCES Diplômé([No diplômé]), [No période sollicitation] INTEGER REFERENCES Sollicitation([No période sollicitation]))

Script de réalisation du modèle en VBA.

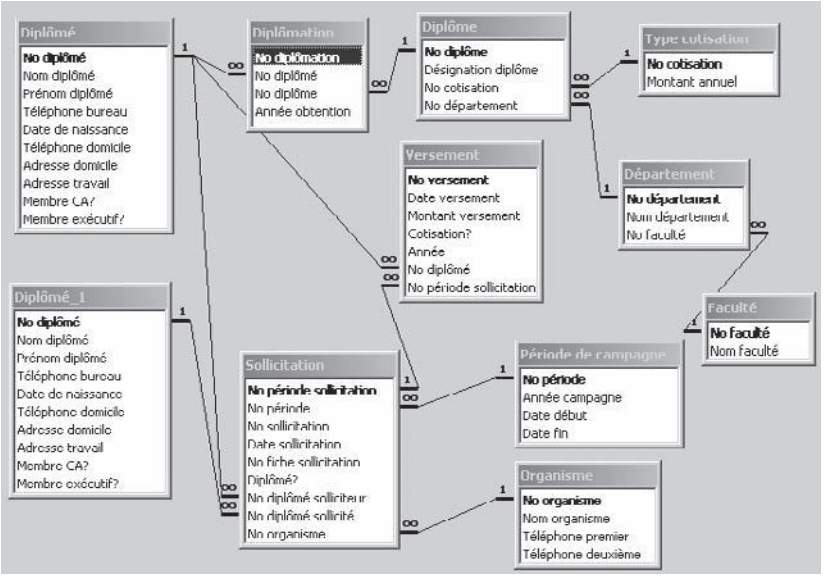
```
Private Sub Script_Creation_Ex3_6()
Dim conDatabase As ADODB.Connection
Dim SQL As String

On Error GoTo Error_Handler
Set conDatabase = Application.CurrentProject.Connection

SQL = _
"CREATE TABLE Diplômé" & _
" ([No diplômé] SMALLINT" & _
" NOT NULL PRIMARY KEY," & _
" [Nom diplômé] VARCHAR(30)," & _
" [Prénom diplômé] VARCHAR(30)," & _
" [Téléphone bureau] VARCHAR(12)," & _
" [Date de naissance] DATE," & _
" [Téléphone domicile] VARCHAR(12)," & _
" [Adresse domicile] VARCHAR(50)," & _
" [Adresse travail] VARCHAR(50)," & _
" [Membre CA?] BIT," & _
" [Membre exécutif?] BIT)"

conDatabase.Execute SQL
MsgBox "Table Diplômé créée", vbInformation

SQL = _
"CREATE TABLE [Type cotisation]" & _
" ([No cotisation] SMALLINT" & _
" NOT NULL PRIMARY KEY," & _
" [Montant annuel] CURRENCY)"

conDatabase.Execute SQL
MsgBox "Table Type cotisation créée", vbInformation

SQL = _
"CREATE TABLE Faculté" & _
```

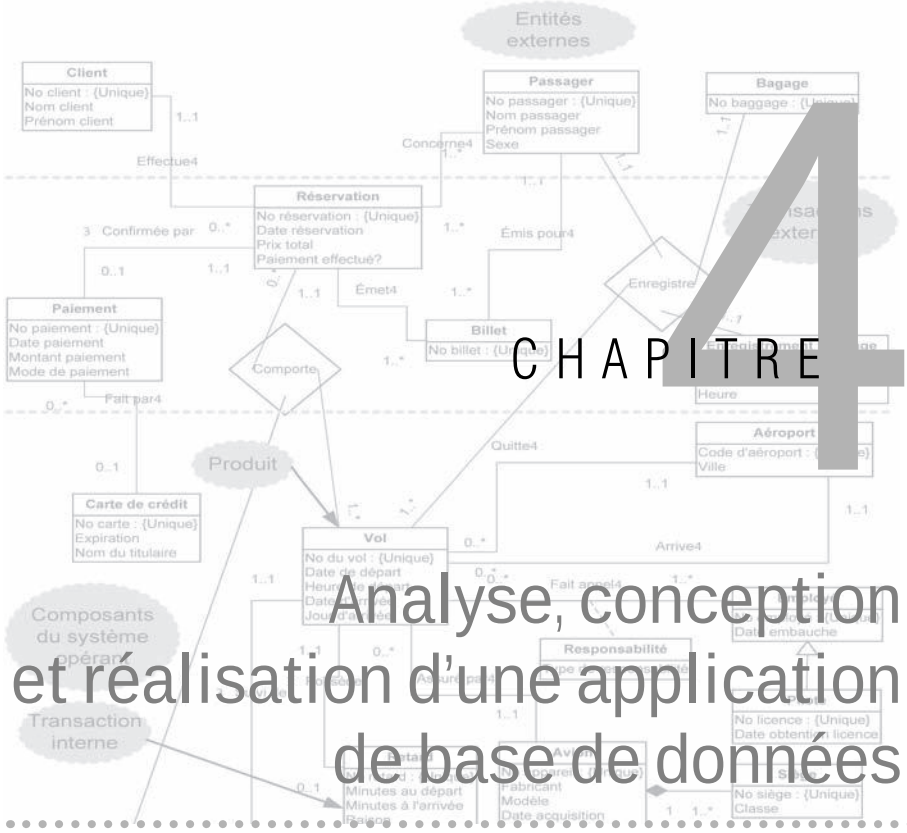
```
" ([No faculté] IDENTITY" & _
" NOT NULL PRIMARY KEY," & _
" [Nom faculté] VARCHAR(30))"

conDatabase.Execute SQL
MsgBox "Table Faculté créée", vbInformation

SQL = _
"CREATE TABLE [Période de campagne]" & _
" ([No période] SMALLINT" & _
" NOT NULL PRIMARY KEY," & _
" [Année campagne] SMALLINT," & _
" [Date début] DATE," & _
" [Date fin] DATE)"

conDatabase.Execute SQL
MsgBox "Table Période de campagne créée", vbInformation

SQL = _
"CREATE TABLE Organisme" & _
" ([No organisme] IDENTITY" & _
" NOT NULL PRIMARY KEY," & _
" [Nom organisme] VARCHAR(30)," & _
" [Téléphone premier] VARCHAR(12)," & _
" [Téléphone deuxième] VARCHAR(12))"

conDatabase.Execute SQL
MsgBox "Table Organisme créée", vbInformation

SQL = _
"CREATE TABLE Département" & _
" ([No département] IDENTITY " & _
" NOT NULL PRIMARY KEY," & _
" [Nom département] VARCHAR(30)," & _
" [No faculté] INTEGER" & _
" NOT NULL" & _
" REFERENCES Faculté([No faculté]))"

conDatabase.Execute SQL
MsgBox "Table Département créée", vbInformation

SQL = _
"CREATE TABLE Diplôme" & _
" ([No diplôme] IDENTITY " & _
" NOT NULL PRIMARY KEY," & _
" [Désignation diplôme] VARCHAR(30)," & _
" [No cotisation] SMALLINT" & _
" NOT NULL" & _
" REFERENCES [Type cotisation]([No cotisation])," & _
" [No département] INTEGER" & _
```

```
" NOT NULL" & _
" REFERENCES Département([No département]))"

conDatabase.Execute SQL
MsgBox "Table Diplôme créée", vbInformation

SQL = _
"CREATE TABLE Sollicitation" & _
" ([No période sollicitation] IDENTITY " & _
" NOT NULL PRIMARY KEY," & _
" [No période] SMALLINT NOT NULL" & _
" REFERENCES [Période de campagne]([No période])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [No sollicitation] SMALLINT," & _
" [Date sollicitation] DATE," & _
" [No fiche sollicitation] SMALLINT," & _
" [Diplômé?] BIT," & _
" [No diplômé solliciteur] SMALLINT" & _
" REFERENCES Diplômé([No diplômé])," & _
" [No diplômé sollicité] SMALLINT" & _
" NOT NULL" & _
" REFERENCES Diplômé([No diplômé])," & _
" [No organisme] INTEGER" & _
" REFERENCES Organisme([No organisme]))"

conDatabase.Execute SQL
MsgBox "Table Sollicitation créée", vbInformation

SQL = _
"CREATE TABLE Diplômation" & _
" ([No diplômation] IDENTITY" & _
" NOT NULL PRIMARY KEY," & _
" [No diplômé] SMALLINT NOT NULL" & _
" REFERENCES Diplômé([No diplômé])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [No diplôme] INTEGER NOT NULL" & _
" REFERENCES Diplôme([No diplôme])" & _
" ON DELETE CASCADE" & _
" ON UPDATE CASCADE," & _
" [Année obtention] DATE))"

conDatabase.Execute SQL
MsgBox "Table Diplômation créée", vbInformation

SQL = _
"CREATE TABLE Versement" & _
" ([No versement] SMALLINT" & _
" NOT NULL PRIMARY KEY," & _
```

```
" [Date versement] DATE," & _
" [Montant versement] CURRENCY," & _
" [Cotisation?] BIT," & _
" [Année] SMALLINT," & _
" [No diplômé] SMALLINT" & _
" REFERENCES Diplômé([No diplômé])," & _
" [No période sollicitation] INTEGER" & _
" REFERENCES Sollicitation([No période sollicitation]))"

conDatabase.Execute SQL
MsgBox "Table Versement créée", vbInformation

conDatabase.Close
Set conDatabase = Nothing
Exit Sub
Error_Handler:
 MsgBox Err.Description, vbInformation
End Sub
```

Schéma des associations dans MS Access.





CHAPITRE 4

Analyse, conception et réalisation d'une application de base de données

OBJECTIFS

- Comment aborder un projet de développement d'une application de base de données en suivant une méthode systématique.
- Principes sous-jacents à la méthode proposée et identification des données persistantes dès la première phase de la méthode.
- Description de la phase d'***analyse*** et formulation des besoins en matière de données persistantes pour l'application.
- Découpage de l'application en unités fonctionnelles selon le formalisme appelé *Diagramme de cas d'utilisation* et recensement des données persistantes.
- Description de la phase de ***conception et de réalisation***.

Au fil des chapitres précédents, nous avons étudié les trois types de modèles de données qui mènent à la réalisation d'une base de données. Nous avons vu qu'ils s'inscrivent dans un continuum dont la cohérence est assurée par un certain nombre de règles de passage d'un niveau de modélisation à l'autre. Nous allons maintenant nous attarder à l'intégration de la démarche de modélisation des données dans un processus plus large visant la réalisation complète d'une application de base de données.

La réalisation des divers modèles abordés aux chapitre précédents, et visant à concevoir une base de données, n'est pas une fin en soi. Cette démarche s'inscrit habituellement dans le cadre d'un projet de mise en œuvre d'une application informatique permettant de saisir, de traiter, de stocker et d'exploiter les données nécessaires au bon fonctionnement d'une organisation ou d'un secteur spécifique d'une organisation. Dans notre introduction, nous avons défini une telle application, qui exploite une ou des bases de données, sous l'appellation ***application de base de données***. Une application de base de données doit être réalisée en suivant une méthode qui permette non seulement l'élaboration de modèles qui traduisent les besoins des utilisateurs en matière de données mais aussi de modèles d'une autre nature exprimant les besoins généraux de ces derniers envers le fonctionnement de l'application.

Ces besoins généraux concernent entre autres les exigences quant à l'interface utilisateur de l'application et sa logique applicative. C'est dans la logique applicative que s'expriment notamment les règles de gestion de certains processus ou de certaines activités de l'organisation. Cette logique applicative doit être en phase avec la structure de la BD.

Le chapitre a pour objectif premier de présenter une méthode concrète, quoique simplifiée, pour mener à bien la réalisation d'une BD et de l'application qui saura l'exploiter efficacement. La méthode est fortement structurée. Elle fait appel aux règles de modélisation des données abordées aux chapitres précédents auxquelles s'ajoutent des règles et principes de définition des exigences portant sur d'autres aspects de l'application. La méthode doit encadrer le travail du modélisateur pour recenser dans une organisation les données nécessaires à son bon fonctionnement et mettre en œuvre la logique applicative qui doit assurer notamment la saisie, la validation et la persistance de données cohérentes.

La méthode préconisée dans cet ouvrage comporte deux phases :

1. la phase d'analyse des besoins, tournée vers l'identification des besoins en matière de données et des exigences de l'application qui en assure l'exploitation ;

2. la phase de conception et de réalisation de l'application, centrée sur les aspects techniques de l'application qui devra répondre efficacement aux besoins et exigences formulées dans la phase d'analyse.

Notre méthode s'inspire des méthodes dites *orientées objets* et en particulier de la méthode du *Processus unifié* [JBR 99]qui suggère une approche légère et itérative pour le développement d'une application informatique. Elle emprunte de plus à la méthode DATARUN [PAS 90] certains principes permettant de guider le modélisateur, au cours de la phase d'analyse, pour le recensement efficace des données persistantes. Chaque phase de la méthode mène à la production de modèles dont certains ont fait l'objet d'une étude approfondie au cours des chapitres précédents. Au cours de la phase d'analyse des besoins, la tâche du modélisateur porte surtout sur le modèle conceptuel de données. Toutefois les modèles logique et physique de données résultent de la deuxième phase, la phase de conception et de réalisation.

Voyons maintenant plus en détail la nature exacte de chaque phase de la méthode.

L'ANALYSE DES BESOINS

Cette section porte sur la démarche de formulation des besoins des utilisateurs en regard d'une application de base de données. Le terme *besoins* s'inscrit ici dans un contexte bien particulier. Aussi adopterons-nous la définition technique donnée ci-après.

Besoins ▸ Conditions auxquelles un système et plus généralement un projet doit répondre.

La phase d'analyse représente un travail systématique de recherche, de documentation, d'organisation et de suivi des besoins des parties intéressées sachant que les souhaits et les besoins exprimés ne sont pas toujours explicites et peuvent changer dans le temps. Il est donc futile de viser l'exhaustivité. Le modélisateur peut difficilement identifier et définir tous les besoins avant de s'engager dans la réalisation de l'application. La méthode que nous proposons est beaucoup plus réaliste. Elle consiste à rechercher les besoins les plus immédiats, ceux qui font consensus entre les parties intéressées, de procéder rapidement à la conception et la réalisation d'une première version de l'application qui en découle, quitte à affiner les besoins connus et à en ajouter de nouveaux au cours des cycles d'analyse ultérieurs.

L'objectif premier de la phase d'analyse des besoins est de définir les besoins des utilisateurs en matière de données. Cette tâche consiste à identifier les données présentes dans une organisation ou un secteur de l'organisation dont il est nécessaire d'assurer la *persistance* et de les modéliser sous forme de diagramme entité-association. Cela revient à dire que le modélisateur doit identifier les données qui seront conservées dans la BD de l'application de manière à répondre aux besoins exprimés par les utilisateurs. Le modèle conceptuel de données réalisé au cours de cette phase constitue la représentation graphique des *données persistantes* de l'application.

Donnée persistante

Une donnée générée dans le cadre des activités d'une organisation ou provenant de l'extérieur de cette organisation qu'il est nécessaire de conserver à long terme pour assurer le bon fonctionnement de l'organisation. Cette définition oppose le concept de donnée persistante à celui de donnée **volatile** qui, elle, disparaît peu de temps après avoir été générée ou reçue sans que cela n'ait de conséquence pour l'organisation (*Persistent Data*).

La notion de données persistante correspond à une notion équivalente introduite dans la méthode DATARUN, celle de *donnée vitale*, notion dont nous avons déjà fait état. En vertu de cette méthode, l'effort du modélisateur au stade de l'analyse des besoins doit porter en priorité sur le recensement des données que l'organisation ne peut se permettre d'ignorer ou de ne pas mémoriser au risque de compromettre sa mission et son bon fonctionnement. On qualifie ces données de *vitales* car elles sont réputées vitales à l'organisation notamment à des fins de contrôle et de support à la décision.

Le recensement des données persistantes, ou vitales, ne peut se faire qu'en suivant une démarche systématique appuyant une réflexion rigoureuse à laquelle prend part non seulement le modélisateur mais aussi les utilisateurs ou les clients du système à réaliser. L'identification d'une donnée persistante est avant tout une affaire de jugement. Elle dépend fortement du contexte et du domaine où s'exerce le travail d'analyse des besoins par le modélisateur. Ce jugement peut s'exercer en suivant certaines lignes directrices. En voici deux. Elles représentent à notre avis des conditions nécessaires mais elles ne sauraient en aucun cas être considérées comme suffisantes.

Une donnée est considérée comme persistante si elle ne peut être dérivée d'une autre donnée persistante connue. Si nous avons identifié la *date de naissance de l'employé* comme une donnée persistante, la donnée *âge de l'employé* n'a pas à être persistante puisque nous pouvons facilement dériver sa valeur de la première et ce en tout temps. Il est donc inutile de conserver dans la base de données ces deux données à la fois car l'une peut être facilement calculée à partir de l'autre. Cette règle n'est pas absolue. Considérons le cas d'une commande pour laquelle on considère

comme données persistantes le *prix* de l'article commandé ainsi que sa *quantité*. On pourrait en conclure qu'il est inutile de considérer le *total* de la commande comme une donnée persistante car on peut le calculer en faisant la somme des produits du prix par la quantité de l'article à laquelle les taxes sont ajoutées. L'utilisateur du système peut voir les choses autrement. Pour ce dernier le total de la commande est une donnée à caractère contractuel qui doit être strictement conforme à la facture remise au client qui a passé la commande. La donnée *total de la commande* devrait dans ce contexte être mémorisée pour chaque commande, telle qu'elle apparaît sur la facture, sans égard à la manière dont elle a été calculée. Il se peut en effet que la formule de calcul change dans le temps car les taux des taxes applicables sont susceptibles d'être modifiés d'une année à l'autre.

Une donnée est persistante si elle est déterminante pour le bon fonctionnement de l'organisation. La *couleur des yeux de l'employé* n'est pas une donnée déterminante pour une organisation. Il ne s'agit donc pas d'une donnée qui devrait être considérée comme persistante pour les organisations en général. Qu'en est-il d'une agence de mannequins où cette donnée peut être vue comme vitale à son fonctionnement ? Cette ligne directrice doit être appliquée en tenant compte du contexte où elle s'exerce. Une organisation pourrait conclure qu'une donnée particulière sur ses clients est déterminante pour son fonctionnement bien que l'exigence faite au client de fournir cette donnée et sa mémorisation pourraient contrevenir à la Loi sur les renseignements personnels des citoyens d'un État. Il serait en conséquence illégal d'en faire une donnée persistante.

S'il est impossible de prescrire des règles absolues pour l'identification des données persistantes propres à une organisation ou à un domaine d'activité, on peut néanmoins proposer une démarche systématique facilitant leur recensement. Notre expérience de modélisateur nous a permis de constater qu'il existe deux approches complémentaires pour recenser les données sur lesquelles le modélisateur devra exercer en priorité son jugement, avec le concours des utilisateurs, pour en faire le cas échéant des données persistantes.

Le recensement des données persistantes

Les deux formes que peut prendre la démarche de recensement des données persistantes découlent d'un certain nombre de constats qui justifient leur pertinence. Avant d'étudier chaque forme que la démarche peut prendre, attardons-nous à quelques-uns de ces constats.

- Bien qu'une organisation ou un secteur d'activité de l'organisation ne soient pas dotés d'un système d'information formel, on peut y retrouver des pratiques de gestion, des mécanismes de contrôle, des protocoles, des règles de gestion qui constituent les éléments d'un système d'information *informel* mais bien réel ;
- Ce système d'information informel fait généralement appel à des documents, des bordereaux, des fiches, des dossiers assurant en quelque sorte une forme de *mémoire* des données du système informel ;
- De tels documents représentent des sources privilégiées de données persistantes pour une application de base de données qui sera réalisée dans le but d'automatiser en tout ou en partie le système d'information informel ;
- S'il s'avère qu'on ne peut compter sur un système d'information existant formel ou informel ou que l'application de base de données à réaliser repose sur des données originales dont ne dispose pas encore l'organisation à travers ses documents imprimés ou électroniques, il est possible d'élaborer de concert avec les utilisateurs un *prototype visuel* de l'application à réaliser ;
- Un prototype visuel est le pendant pour une application informatique du *story-board* dessiné par le réalisateur d'un film et son scénariste avant de procéder au tournage ;
- Le prototype visuel est constitué d'une série de dessins, le plus souvent sur papier, des panoramas d'écran qui correspondent à une ébauche de l'interface utilisateur de l'application ;
- Chaque panorama est une source privilégiée de données persistantes pour une application de base de données en absence de système d'information formel ou informel.
- Les dessins de ces panoramas peuvent facilement être réalisés par affinage progressif lors de séances de remue-méninges (*brainstorming*) impliquant les utilisateurs et les concepteurs.

Les constats évoqués ici sont à la base des deux approches préconisées pour le recensement des données persistantes. La première approche est applicable à une situation où un système d'information, même informel ou embryonnaire, est déjà en place dans l'organisation. La seconde vise la réalisation d'une application originale, sans équivalent formel ou informel, où les utilisateurs expriment leurs besoins principalement à travers un projet d'interface utilisateur, élaborée à la manière d'un story-board au cinéma, appelé *prototype visuel de l'application de base de données*. Les deux approches peuvent être appliquées concurremment pour la définition des besoins des

utilisateurs tout en sachant que l'application en matière de données repose à la fois sur un système d'information existant et sur des besoins originaux non assurés par le système en place.

La première approche pourrait être qualifiée d'approche *descendante* puisqu'elle débute par une analyse d'un système existant pour modéliser en détail le nouveau système à réaliser. L'autre approche est *ascendante*. L'analyse débute en élaborant une représentation visuelle de l'interface utilisateur du système que l'on souhaite réaliser, pour en tirer ensuite les modèles nécessaires à sa réalisation.

Nous allons aborder, à tour de rôle, chacune des approches et les illustrer par des études de cas. Ces deux approches de la phase d'analyse, bien que différentes au début, mènent au même résultat : la production d'un modèle conceptuel pour les données persistantes de l'application de base de données que l'on souhaite réaliser. D'aucune manière, elles ne remettent en cause les règles et les principes de modélisation conceptuelle décrits au chapitre 1. Bien au contraire. Elles permettent d'identifier et d'inventorier en amont les données pertinentes au modèle conceptuel de données qui conduira ultérieurement, durant la phase de conception et de réalisation, à la production des modèles logique et physique de la base de données.

Une approche descendante : le modèle de fonctionnement du système d'information

Cette première approche consiste à faire l'analyse d'un système d'information existant, formel ou informel, et à en tirer les besoins en matière de données pour automatiser certains aspects de son fonctionnement à l'aide d'une application de base de données.

La première difficulté de cette approche est de formuler une représentation abstraite du fonctionnement de ce système d'information avant d'aborder la modélisation conceptuelle des données persistantes du système. Cette représentation abstraite du système d'information s'appelle un *modèle de fonctionnement du système d'information*. Elle fait appel à un formalisme nouveau appelé *Diagramme de cas d'utilisation*. La notation UML, qui permet d'élaborer de tels modèles, est conforme à celle introduite dans les méthodes d'analyse dites orientées objets qui ont donné naissance au formalisme.

Diagramme de cas d'utilisation ▸ Formalisme qui permet de modéliser le fonctionnement d'un système par un découpage de celui-ci en fonctionnalités. Il illustre de plus la nature des interactions avec ces fonctionnalités, offertes à titre de services à des acteurs externes au système. Le découpage peut être global ou très éclaté. La nature des interactions peut être décrite de manière sommaire ou détaillée selon le niveau de détail recherché par le modélisateur. Chaque fonctionnalité est appelé un **cas d'utilisation** (*Use Case Diagram*).

Un diagramme de cas d'utilisation comporte une composante graphique, où sont illustrés les fonctionnalités du système et les acteurs qui interagissent avec elles. L'autre composante du diagramme de cas d'utilisation, tout aussi importante, est la documentation textuelle de chaque fonctionnalité sous une forme que nous appellerons *scénario de cas d'utilisation*. La composante graphique permet d'avoir une vue d'ensemble des fonctionnalités d'un système, de situer la frontière entre le système et les acteurs extérieurs qui gravitent autour de celui-ci. Elle illustre de plus les fonctionnalités du système auxquelles un acteur en particulier peut faire appel.

La composante textuelle décrit sous forme de scénarios, pour chaque fonctionnalité, comment se déroule chronologiquement une séance d'interaction entre un acteur (par exemple un utilisateur du système d'information qui requiert des services de celui-ci) et une fonctionnalité du système. Elle stipule notamment comment est amorcée une séance, quelles sont les options dont dispose l'acteur pour interagir avec la fonctionnalité et quelles sont les conditions pour terminer une séance avec succès.

Étudions maintenant de façon plus détaillée chaque composante du diagramme de cas d'utilisation. Désormais, pour rester en phase avec la terminologie utilisée généralement pour traiter des cas d'utilisation, nous utiliserons le terme *cas d'utilisation* au lieu de *fonctionnalité*, et *scénario de cas d'utilisation* pour faire référence à la description textuelle du cas d'utilisation.

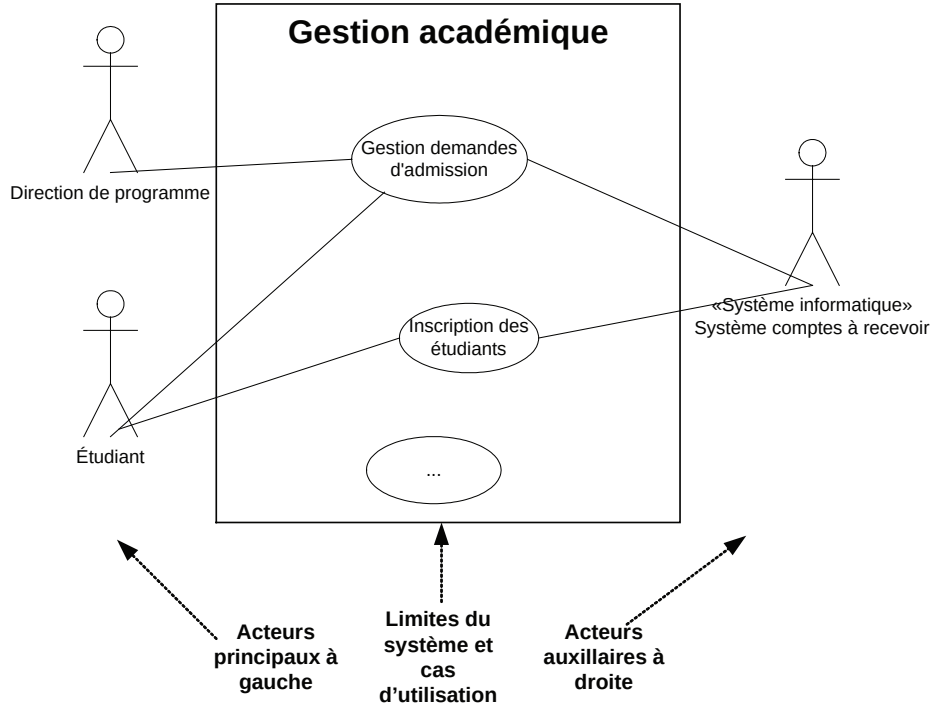
Le côté graphique d'un diagramme de cas d'utilisation

La représentation graphique des cas d'utilisation fait appel à quatre symboles élémentaires :

1. l'acteur (petit bonhomme),
2. le cas d'utilisation (losange), correspondant à une fonctionnalité du système
3. les limites du système (grand rectangle entourant les cas d'utilisation)
4. l'arc ou segment de droite liant l'acteur à un cas d'utilisation.

La figure 4-1 montre un diagramme de cas d'utilisation incomplet qui nous servira d'exemple tout au long de la démonstration suivante et qui portera sur le rôle joué par chacun de ces symboles dans la modélisation du fonctionnement d'un système.

FIGURE 4-1 **Diagramme de cas d'utilisation incomplet modélisant un système d'information pour la *Gestion académique d'un Collège***



Tout diagramme de cas d'utilisation porte une identification du système modélisé, ici **Gestion académique**. Dans ce cas, il s'agit d'un système d'information qui n'est pas informatisé mais fait appel à un système informatisé pour la gestion des comptes à recevoir des étudiants. Les limites du système tracent une frontière entre les acteurs extérieurs au système et les fonctionnalités du système désormais appelées ***cas d'utilisation***. Notons au passage qu'un diagramme de cas d'utilisation peut aussi être utilisé pour modéliser un système informatisé ou un logiciel. Dans la méthode descendante, préconisée dans cette section, nous en faisons usage pour la modélisation d'un

système d'information existant, qu'il soit formel ou informel. À la section suivante nous verrons comment l'exploiter pour modéliser le fonctionnement d'une application de base de données, selon une approche ascendante.

Acteur ▸ Une entité quelconque ayant un comportement qui inclut d'autres systèmes faisant appel au système modélisé ou auquel le système modélisé fait appel. Plus généralement, il s'agit d'une catégorie de personnes qui peut agir comme utilisateur du système et interagir avec ce dernier (*Actor*).

La notion d'acteur ne doit pas être vue comme trop restrictive. Elle englobe des acteurs dit ***principaux***, soit des utilisateurs qui ont des buts bien précis à atteindre en utilisant les services du système. Dans l'exemple de la figure 4-1, **Étudiant** est un acteur de ce type. Il fait appel à un service du système pour que soit traitée sa demande d'admission au Collège. Par convention, les acteurs principaux sont disposés à gauche dans un diagramme de cas d'utilisation.

Pourquoi identifier les acteurs principaux ? Pour pourvoir identifier les services que leur livre le système et les données générées lors de la prestation de ces services.

Un acteur peut être ***auxiliaire***. Il s'agit ici d'un acteur qui *fournit* un service au système modélisé, au contraire de l'acteur principal qui les sollicite. Ce service peut être une donnée où un ensemble de données dont le système a besoin pour répondre à un service sollicité par un acteur principal. L'acteur auxiliaire est souvent un autre système, notamment un système informatisé, mais il peut aussi s'agir d'une autre organisation ou même d'une personne externe au système que l'on modélise et qui agit comme prestataire de services auprès du système.

Pourquoi identifier les acteurs auxiliaires ? Pour pourvoir identifier les données acquises par le système de sa propre initiative. Cette acquisition peut se faire par d'autres moyens qu'un formulaire ou la saisie électronique de données via un dispositif de saisie tel qu'un clavier ou un écran sensible. Ce seront souvent des données acquises directement qui exigeront la mise en œuvre d'interfaces logicielles avec d'autres systèmes lors de la réalisation de l'application de base de données.

Cas d'utilisation ▸ Une collection de scénarios qui décrit la façon dont un ou plusieurs acteurs interagissent avec un système pour atteindre un but. Les scénarios prennent la forme d'une description textuelle d'actions ou d'interactions entre un ou des acteurs et le système modélisé (*Use Case*).

Dans le diagramme de cas d'utilisation 4-1, on note la présence de l'acteur **Étudiant**, susceptible de s'engager dans une interaction avec **Gestion demandes d'admission**. Ce cas d'utilisation fait intervenir un autre acteur soit une **Direction de programme** responsable d'accepter où de refuser une demande d'admission faite par l'Étudiant pour un programme d'étude dont elle a la responsabilité.

À la lecture des scénarios du cas d'utilisation **Gestion demandes d'admission**, le lecteur comprendra que l'acteur **Direction de programme** est aussi utilisateur du système. En effet, il fera appel périodiquement à ses services pour obtenir des informations sur les étudiants qui ont soumis une demande d'admission dans un des programmes sous sa responsabilité, étudiants qui ont de plus été admis et qui ont ensuite accepté l'offre d'admission.

Considérons maintenant en quoi consiste les scénarios associés à chacun des cas d'utilisation.

Les scénarios associés au cas d'utilisation **Gestion demandes d'admission** sont donnés dans le tableau 4-1. Cet exemple présente une description *abrégée* du cas d'utilisation. Il s'agit d'un résumé qui montre un scénario de base et quelques scénarios alternatifs pour tenir compte de cas particuliers ou de situations d'exception.

Aux fins de modéliser le *fonctionnement d'un système d'information*, il n'y a pas lieu de scénariser les moindres détails d'un cas d'utilisation. Dans l'exemple du tableau 4-1, il est inutile de spécifier en détail comment procède le Bureau du registraire pour attribuer un code à l'Étudiant ou pour créer un compte dans le système des comptes à recevoir. Il est par ailleurs très important D'IDENTIFIER LES DOCUMENTS QUI SERVENT DE SUPPORT AUX DONNÉES exploitées dans le cadre des activités scénarisées dans le cas d'utilisation. Notamment les documents papiers. De tels documents sont la source privilégiée pour recenser les données dont le système d'information doit assurer la persistance.

Sous cette rubrique, nous tentons de montrer comment modéliser le fonctionnement d'un système d'information existant afin de formuler les besoins en matière de données pour une application de base de données assurant la persistance des données du système d'information.

Les objectifs de la démarche de représentation du fonctionnement d'un système d'information, fut-il formel ou informel, sous forme de diagramme de cas d'utilisation, peuvent se résumer ainsi :

1. Le modèle de fonctionnement du système d'information constitue une représentation graphique du ***contexte*** de ce système, donc de son environnement ;

TABLEAU 4-1 **Scénarios du cas d'utilisation Gérer demande d'admission**

Scénarios	Cas d'utilisation : *Gestion des demandes d'admission*
Principal	L'Étudiant complète le formulaire **Demande d'admission**, y joint les documents exigés tels qu'une copie de ses attestations d'études et un chèque libellé au nom du Collège couvrant les frais d'admission. L'Étudiant transmet le formulaire complété et les documents requis au Bureau du registraire du Collège. Le Bureau du registraire reçoit la demande et procède à la vérification des documents reçus. Si ces derniers sont conformes et complets, le Bureau attribut un code à l'Étudiant et ouvre un compte dans le système des comptes à recevoir si l'Étudiant n'en a pas déjà un. Il encaisse le chèque et ne crédite le compte du montant versé que si le chèque couvre la totalité des frais d'admission. Si l'Étudiant ne dispose d'aucun solde de compte à payer à son dossier, le Bureau transmet son dossier de candidature à chaque Direction de programme pour lequel l'Étudiant soumet une demande d'admission. À chaque dossier de candidature est joint le formulaire **Décision de la direction du programme**. La Direction du programme reçoit le dossier, en fait l'étude et complète le formulaire **Décision de la direction du programme** où elle inscrira sa décision et en cas de refus, les raisons qui ont conduit au refus. Le Bureau reçoit la où les décisions et en informe l'Étudiant en lui transmettant le formulaire **Décision d'admission**. L'Étudiant accepté dans un ou plusieurs programmes doit retourner ce formulaire au Bureau du registraire en confirmant son acceptation d'admission pour un seul programme où il est admis. Le Bureau reçoit le formulaire et inscrit la confirmation de l'Étudiant à son dossier pour le programme mentionné.
Alternatif : Les documents reçus de l'Étudiant ne sont pas conformes	Le Bureau du registraire informe l'Étudiant de la nature des corrections à apporter à sa demande pour qu'elle soit conforme. Le Bureau complète le formulaire **Avis de non conformité de la demande d'admission** pour informer l'Étudiant des raisons de non conformité (Ex. Chèque sans provision ou non remis, Attestations incomplètes, etc.). L'Étudiant transmet les documents exigés pour assurer la poursuite du traitement de sa demande.
Alternatif : Une Direction de programme souhaite un état des Étudiants qui ont accepté leur admission	La Direction de programme complète un formulaire **Demande d'état des candidatures confirmées** où elle inscrit la session et les programmes pour lesquels elle souhaite recevoir un état à jour des candidatures confirmées. Elle le transmet au Bureau du registraire. Le Bureau produit cet état et le transmet à la Direction de programme.

- il permet de tracer les frontières du système pour circonscrire les sources de données externes et les cas d'utilisation susceptibles de les exploiter ou d'en générer de nouvelles dont on assurera la persistance par le biais d'une base de données ;
- il permet d'inventorier les formulaires et les documents qui constituent la source principale de données persistantes du système ;
- il est à l'origine de la réalisation d'un modèle conceptuel de données produit après le recensement des données persistantes présentes dans les formulaires et les documents ou dérivé des scénarios de cas d'utilisation.

2. Les cas d'utilisation doivent assurer une représentation de ce que le système fait et donc de ses fonctionnalités telles que vues par les acteurs (personnes ou systèmes externes au système) ;
   - ils décrivent le comportement du système en regard d'activités logiquement regroupées : par exemple la gestion des admissions dans le collège, la gestion des inscriptions à chaque session, le processus d'émission des diplômes ;
   - ils sont axés sur les fonctions du système d'information et sur les données qu'ils consomment ou qu'ils génèrent.
3. Les cas d'utilisation doivent permettre d'assurer le découpage fonctionnel du système d'information de manière à permettre la planification du processus d'analyse, de conception et de réalisation d'une application de base de données ;
   - le découpage fonctionnel devrait faciliter la planification des phases successives de réalisation de l'application assurant la persistance des données pour le système ;
   - à l'aide d'un tel découpage, la planification consistera à produire un ordonnancement chronologique des cas d'utilisation (donc des fonctionnalités) à mettre en œuvre dans l'application de base de données.

Difficultés rencontrées dans la réalisation du modèle de fonctionnement d'un système d'information

La production du modèle de fonctionnement d'un système d'information exige de la part du modélisateur une grande capacité de synthèse. Il pourrait se voir confronté à trois grandes difficultés :

1. choisir les limites du système ;

2. distinguer les acteurs principaux, les acteurs auxiliaires et les personnes ou les groupes de personnes qui ne sont pas acteurs mais qui font partie intégrante du système d'information ;
3. identifier les cas d'utilisation en tentant de regrouper les activités du système par blocs, chaque bloc d'activités logiquement apparentées représentant une fonctionnalité de haut niveau, pour les rattacher ensuite aux acteurs.

Nous proposons de produire le modèle en attaquant les difficultés dans cet ordre.

Établir en premier lieu les limites du système. Comment choisir les limites du système ? D'abord en tentant de donner au système à modéliser un nom pertinent. L'appellation **Gestion académique** dit bien ce que ce système d'information supporte : les activités de gestion qui touchent aux aspects académiques exclusivement. Il n'assure pas la gestion comptable ou financière du collège, ni la gestion de ses terrains, bâtiments et équipements. Une appellation appropriée est un bon point de départ pour circonscrire les frontières du système à modéliser.

On tente d'identifier ensuite les agents extérieurs au système qui traitent avec lui, soit les acteurs. On prendra en considération les personnes d'abord. Quelles sont les personnes qui sont les *déclencheurs* de services offerts par le système ou en sont les *bénéficiaires* et qui ne sont pas parties prenantes dans les opérations du système d'information ? Rappelons qu'un système d'information est constitué de procédures, de documents, de personnes, de matériels et de logiciels qui supportent ses opérations. Le bureau du registraire représente un groupe de personnes qui font fonctionner en tout ou en partie le système de gestion académique. Il ne peut en aucun cas être vu comme externe au système ; il en est une composante essentielle. On dira plutôt que le bureau du registraire est une *ressource* du système d'information, tout comme les procédures et la réglementation qui dictent les règles de gestion administrative du collège ou, à un niveau de moindre importance, les machines à écrire qui servent à compléter le formulaire **Décision d'admission**. Le bureau du registraire ne déclenche aucun service commandé de l'extérieur du système, ce n'est donc pas un acteur (sous-entendu externe au système). C'est l'étudiant qui, en adressant une demande d'admission au collège, déclenche un service et en devient le bénéficiaire. Voilà un véritable acteur au sens où on l'entend pour la rédaction des cas d'utilisation. La direction d'un programme intervient certes dans la gestion des admissions comme le montre le cas d'utilisation où son rôle consiste à sanctionner ou non une candidature. Son rôle dans le fonctionnement du système d'information pourrait en faire à cet égard une *ressource*. Mais elle agit aussi comme acteur

car elle requiert de plus des services au système en demandant périodiquement la production d'états qui lui sont remis par le bureau du registraire. À ce titre, elle est aussi un acteur.

Quelles sont les organisations ou les autres systèmes qui peuvent *déclencher* des services offerts par le système ou en être *bénéficiaires* ? Quels sont les organisations ou les autres systèmes qui peuvent agir comme *fournisseurs* de données ou *prestataires de services* à la demande pour que le système d'information puisse répondre à un service déclenché par un autre acteur. Le système de gestion des comptes à recevoir est nettement un acteur car il agit à titre de prestataire de service, pour indiquer que l'étudiant est en dette ou non envers le collège. Il est nettement à l'extérieur des frontières du système d'information car on fait complètement abstraction des opérations internes de ce système pour modéliser le fonctionnement du système de Gestion académique.

Distinguer en deuxième lieu les acteurs principaux et les acteurs auxiliaires. Si le choix des acteurs a été fait correctement, on aura évité d'assimiler un acteur à une ressource interne du système d'information à modéliser. Un acteur principal est un *déclencheur de services* pour le système à modéliser ou un *bénéficiaire* de services. Il peut s'agir d'une personne, un groupe de personnes, d'une organisation ou d'un autre système. Ce qui intéresse le modélisateur, c'est la nature des données transmises par l'acteur pour déclencher le service requis ainsi que les échanges de données subséquents qui vont assurer la prestation du service à cet acteur principal.

Un acteur auxiliaire est un *prestataire* ***externe*** *de services* auquel le système peut faire appel pour répondre à un service déclenché généralement par un acteur principal. Son fonctionnement interne est sans intérêt pour le modélisateur. Ce qui intéresse le modélisateur c'est la nature des données que le système échange avec ce prestataire et non pas la manière dont le prestataire les a générées au bénéfice du système d'information.

Élaborer en dernier lieu les cas d'utilisation. Considérer à tour de rôle chaque acteur principal. À ce titre il déclenche un ou des services qui sont mis en œuvre à travers un ou plusieurs cas d'utilisation et/ou il est le bénéficiaire de services que les ressources internes du système ont déclenchés. On s'intéressa d'abord aux services déclenchés par un acteur principal. Les services de nature différente déclenchés par des événements différents devraient donner lieu à des cas d'utilisation distincts pour le même acteur principal, surtout si l'un précède l'autre chronologiquement. Par exemple le cas **Gestion demandes d'admission** est déclenché par l'étudiant en soumettant sa demande d'admission. Si l'étudiant confirme son admission et s'inscrit ultérieurement, cette inscription sera prise en charge par un second cas d'utilisation, **Inscription des étudiants**, car ce dernier est déclenché par un

Un acteur peut être à la fois principal dans un cas d'utilisation et auxiliaire pour un autre cas. Dans une telle situation, on le considérera avant tout comme acteur principal. Considérons un système d'information appelé **Gestion des approvisionnements**. Ce système possède deux cas d'utilisation : **Produire les commandes aux fournisseurs** et **Traiter la réception de marchandises**. L'acteur Fournisseur est nettement un prestataire de services car il traitera les commandes produites par notre système. Cela en fait un acteur auxiliaire. Par ailleurs, lors de la réception des marchandises qu'il nous livre, il déclenche dans notre système les services de mise à jour de l'inventaire et de paiement aux fournisseurs. Cette dernière activité en fait aussi un acteur principal. Notons enfin que les services d'un système d'information ne sont pas toujours déclenchés par un acteur principal. Certains services SONT DÉCLENCHÉS PAR LES RESSOURCES INTERNES du système d'information. Par exemple, les commandes aux fournisseurs sont émises si le niveau de stock de certains produits atteint le point de rupture, sans intervention d'acteur, donc d'agent externe. Par convention, on dispose sur le diagramme de cas d'utilisation les acteurs principaux à gauche et les acteurs auxiliaires à droite.

événement différent : l'inscription de l'étudiant. Il s'agit bien de deux cas d'utilisation déclenchés par la même catégorie d'acteurs, mais par le biais d'événements différents qui se déroulent dans des temps différents. Chaque cas pourrait par ailleurs faire intervenir des ressources de nature différente dans le système d'information.

Dès que le cas d'utilisation déclenché par un acteur principal est identifié, il y a lieu de voir quels sont les acteurs auxiliaires qui interviennent dans ce cas, bien qu'ils puissent agir aussi à titre d'acteur principal dans d'autres cas. Tous les acteurs du cas étant connus, on procède finalement à la rédaction des scénarios du cas.

On terminera la rédaction des cas d'utilisation par ceux qui ne sont pas déclenchés par des acteurs principaux mais bien par des ressources internes au système d'information pour servir un ou des acteurs principaux. On les rattachera, s'il y a lieu, à des acteurs auxiliaires si les services font appel à des prestataires externes.

Il est fort possible, notamment en rédigeant un scénario, que le cas d'utilisation donne lieu à la création de nouveaux acteurs auxiliaires non prévus initialement. Il peut même arriver que l'on découvre de nouveaux acteurs principaux après avoir identifié un certain nombre de cas d'utilisation, car la présence d'un cas d'utilisation peut en induire un autre qui le précède ou le suit chronologiquement dans les processus de gestion de l'entité dont on modélise le système d'information.

Nous avons présenté quelques principes de construction des diagrammes de cas d'utilisation. Nous allons maintenant poursuivre l'étude de la modélisation du fonctionnement d'un système d'information à travers deux études de cas. Pour chaque cas, il nous faudra partir d'une description de certaines activités d'une organisation pour arriver à en tirer un modèle partiel de fonctionnement de son système d'information. Il s'agit d'une vue théorique des choses car, en pratique, on ne planche pas sur un modèle de fonctionnement du système d'information à partir d'une description des activités de l'organisation. Le modélisateur doit plutôt se déplacer dans l'organisation ou le secteur visé par le projet d'automatisation du système d'information, en étudier les processus d'affaires, collecter des exemplaires des documents utilisés dans ces processus, discuter avec les personnes qui traitent les données contenues dans ces documents, comprendre les règles de gestion qui assurent le bon fonctionnement du système d'information, bien assimiler le vocabulaire du domaine ou du métier concerné, pour finalement produire une synthèse de cette cueillette d'informations sous la forme d'un modèle de fonctionnement du système d'information. Le modèle devra par ailleurs être validé avec les parties prenantes au cours de séances de travail qui pourraient donner lieu à des ajustements majeurs où à l'affinage du modèle.

Nos visées étant avant tout didactiques, il est hors de question d'entraîner le lecteur dans toute la complexité du travail d'un analyste ou modélisateur professionnel. Nous préférons proposer une version simplifiée d'une situation et mettre en lumière l'application de la démarche préconisée ainsi que l'utilisation judicieuse des principes et des techniques introduits plus tôt, de manière à produire un modèle qui soit compréhensible et conforme à l'exposé du cas.

CAS 4-1 TANNERIE TANBEC

(DP) À partir d'une description sommaire des activités de la tannerie TANBEC, proposer un modèle de fonctionnement de leur système d'information bien qu'il n'existe pas sur une base formelle. Procéder systématiquement en établissant d'abord les frontières d'un système ne supportant que les activités décrites dans le cas. Identifier ensuite les acteurs principaux et les acteurs auxiliaires. Dessiner le diagramme de cas d'utilisation, et rédiger deux scénarios, un premier déclenché par un acteur, un deuxième déclenché de l'intérieur du système par une ressource interne.

Énoncé du cas :

La tannerie TANBEC est une petite tannerie qui travaille à la demande. Les clients sont des clients réguliers qui apportent directement les peaux à tanner (en spécifiant le type de tannage requis) ou commandent une peau tannée de surface spécifiée sans fournir les peaux. Dans chaque cas, il y a un seul type de tannage par commande. Lorsque le client se présente, le commis rédige une commande à l'aide du formulaire **Commande client** où sont inscrites les informations sur le client (peaux fournies ou non, nombre de peaux, surface brute des peaux fournies, type de tannage requis). Une grille de tarification appelée **Tarification** permet de déterminer un prix par unité de surface tannée livrée, selon que la peau est fournie ou non pour un type de tannage donné. La commande du client comporte de plus une date estimée de livraison. Dès lors que le tannage est terminé, on complète la commande en inscrivant la date de livraison réelle, la surface tannée livrée, et on calcule enfin le coût total de la commande.

Lorsque le client n'apporte pas les peaux, la tannerie doit commander les peaux nécessaires auprès d'un de ses fournisseurs, en tenant compte de la surface nette exigée. On complète alors une **Commande au fournisseur**. Une commande au fournisseur spécifie la surface brute nécessaire d'un type de peau donné et reprend le prix de vente entendu lors d'une conversation téléphonique avec le fournisseur. Un seul type de peau est commandé au fournisseur et cette commande réfère au client pour lequel elle est requise.

La **Facture** au client est émise à la fin du mois pour les commandes complétées et livrées au cours du mois. On fait un suivi du solde du compte du client à l'aide d'un **Dossier client** de façon à ne plus prendre de commandes de sa part si ce solde est supérieur à 100 $ et en souffrance depuis plus de 30 jours. Un **Reçu** est transmis au client sur encaissement d'un paiement sur facture. Par ailleurs, une **Facture de fournisseur** donne lieu à l'envoi au fournisseur d'un paiement complet dans les 30 jours suivant sa réception.

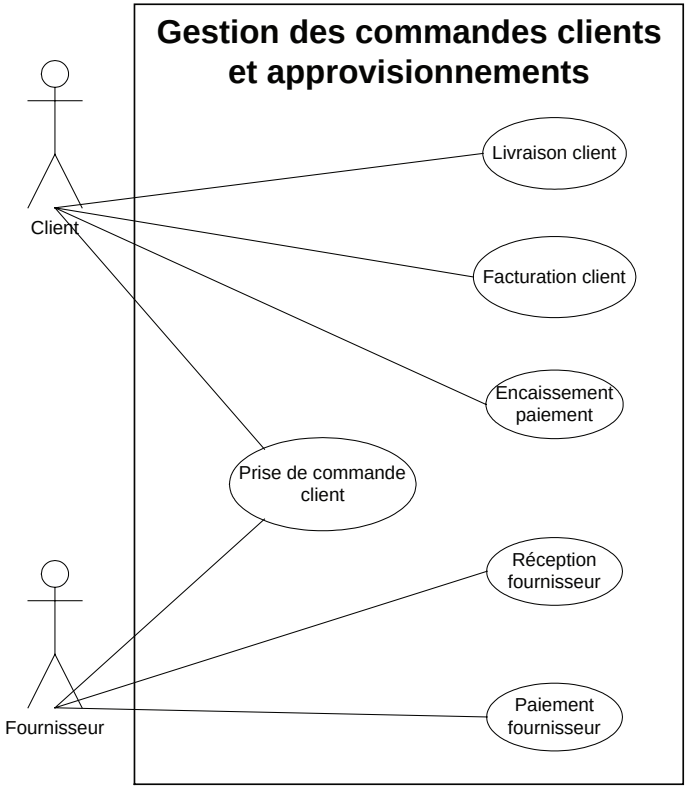
Étude du cas :

Pour bien décrire le système d'information qui supporte les activités de la tannerie TANBEC, l'appellation **Gestion des commandes clients et des approvisionnements** serait la plus appropriée, car le système doit agir comme mémoire à la fois de ce que les clients lui commandent mais aussi des approvisionnements qui découlent de ces commandes.

Deux acteurs se profilent nettement : le client et le fournisseur. Tous les deux déclenchent à tour de rôle des services du système d'information. D'abord le client en soumettant sa commande et le fournisseur en livrant les peaux commandées et en émettant sa facture. Il sont nettement hors des frontières du système. Quant au commis qui complète la commande du client ou qui procède à une commande à un fournisseur, il s'agit bien d'une ressource interne au système d'information et non d'un acteur externe.

Le découpage en cas d'utilisation doit refléter les processus de gestion de la tannerie. Elle reçoit une commande qui peut exiger ou non la prestation d'un fournisseur, elle reçoit du fournisseur le matériel requis, elle livre au client le produit fini. Elle facture son client, paie son fournisseur et encaisse le paiement du client. Chacune de ces activités peut donner lieu à un cas d'utilisation différent car il est déclenché par un événement donné, dont la source est soit un acteur soit une ressource interne au système.

Le diagramme de cas d'utilisation suivant reflète assez fidèlement le fonctionnement du système d'information.



Le cas d'utilisation **Prise de commande client** est le premier qui a été mis en évidence car il fait intervenir les deux acteurs à la fois. Il initie une cascade d'activités qui se succèderont pour assurer la gestion des commandes et approvisionnements. Il met en cause l'acteur Client comme déclencheur et l'acteur Fournisseur comme prestataire de services. Le cas stipule que si une commande de client ne comporte pas de peaux, une commande à un fournisseur doit être faite pour donner suite à la commande du client. Les services de ce cas d'utilisation se déroulent dans un même temps.

Le cas **Réception fournisseur** est déclenché par l'acteur Fournisseur lorsque ce dernier livre les peaux commandées. Il est donc déclenché par un événement autre que celui qui a déclenché la **Prise de commande client** et surtout par un autre acteur. Il s'agit bien de deux cas différents.

Le cas **Livraison client** est déclenché de l'intérieur même du système dès lors qu'une commande client a été exécutée. La **Facturation client** est déclenchée de l'intérieur du système mais dans un temps différent que la **Livraison client**, car le mode d'opération de la tannerie montre clairement que la facturation ne s'effectue qu'à la fin du mois pour tous les clients dont une ou des commandes ont été réalisées au cours du mois qui se termine. Deux événements déclencheurs différents justifient ici deux cas d'utilisation.

Le cas **Encaissement paiement** est déclenché par l'acteur Client lors de la réception de son paiement. Par ailleurs, la réception d'une facture de l'acteur Fournisseur déclenche le cas **Paiement fournisseur**.

Ce n'est pas une règle absolue que de suggérer qu'un événement déclencheur devrait donner lieu à un cas d'utilisation. Tout dépend de la complexité du système à modéliser. Rappelons que le modèle de fonctionnement du système d'information doit donner une vue d'ensemble du contexte d'un système. Un trop grand nombre de cas d'utilisation peut rendre l'exercice futile. Aussi, pour les systèmes de grande envergure, le modélisateur choisira des cas d'utilisation regroupant éventuellement plusieurs services. Chaque cas pourrait alors être déclenché par un événement principal suivi d'événements subsidiaires qui lui sont logiquement liés et pris en charge à l'intérieur du même cas d'utilisation de manière à éviter un trop grand éclatement des cas d'utilisation. C'est ce que nous avons fait pour le système **Gestion académique** avec le cas **Gestion demandes d'admission**. L'événement déclencheur principal du cas est la demande soumise par l'étudiant. Le cas prend par ailleurs en charge des événements subsidiaires tels que la confirmation par l'étudiant de son acceptation ou la demande de production par une direction de programme d'un état des étudiants admis et confirmés. Au delà de dix ou quinze cas d'utilisation, la compréhension d'un modèle pourrait être compromise, notamment à cause du croisement des arcs liant les cas aux acteurs.

Considérons maintenant la rédaction des scénarios des cas d'utilisation. Pour cet exercice, nous retiendrons le cas **Prise de commande client** déclenché par le client et **Facturation client** déclenché de l'intérieur du système.

Chaque scénario principal devrait débuter par l'identification de l'événement principal qui déclenche le cas. Le scénario doit impérativement faire état des documents et des formulaires utilisés dans les activités décrites aux scénarios et qui agissent comme mémoire de données pour le système d'information. Ces documents sont identifiés en caractères gras dans les scénarios.

Scénarios	**Cas d'utilisation :** ***Prise de commande client***
Principal	Le Client se présente pour passer une commande. S'il s'agit d'un client connu, le commis s'assure qu'il n'y a pas au **Dossier client** un compte à payer de plus de 100$ et en souffrance depuis plus de 30 jours. Le commis rédige une commande à l'aide du formulaire **Commande client** où sont inscrites en outre les informations sur le client (peaux fournies ou non, nombre de peaux, surface brute des peaux, type de tannage requis). À l'aide d'une grille de tarification appelée **Tarification** le commis inscrit sur la commande un prix par unité de surface tannée livrée, selon que la peau est fournie ou non pour un type de tannage donné. Il note de plus sur la commande du client une date estimée de livraison.
Alternatif : Le client ne fournit pas les peaux à tanner	Lorsque le client n'apporte pas les peaux, la tannerie doit commander les peaux nécessaires auprès d'un de ses fournisseurs en tenant compte de la surface nette exigée. Le commis doit trouver un fournisseur pour le type de peau requis et pour la surface exigée par le client. Il complète alors une **Commande au fournisseur**. Une commande au fournisseur spécifie la surface brute nécessaire d'un type de peau donné et reprend le prix de vente convenu lors d'une conversation téléphonique avec le fournisseur. Un seul type de peau est commandé par commande au fournisseur et la commande fait référence au client pour lequel elle est requise.

Scénario	**Cas d'utilisation : *Facturation client***
Principal	La facturation des clients est effectuée par un commis le dernier jour du mois. Une **Facture** au client est émise pour toutes les commandes complétées et livrées au cours du mois. Elle est transmise au client. Le commis met à jour au **Dossier client** le solde du compte à recevoir en y incluant la nouvelle facture émise.

Bien qu'incomplet, car les scénarios de cas d'utilisation ne sont pas tous rédigés, le modèle de fonctionnement du système d'information de la tannerie TANBEC répond bien aux objectifs poursuivis avec un tel modèle :

1. Le modèle de fonctionnement du système d'information constitue une représentation graphique du contexte du système et met en évidence les sources de données externes ; il permet d'inventorier les formulaires et les documents qui sont à la source des données persistantes du système.

2. Il assure une représentation de ce que le système fait et donc de ses fonctionnalités telles que vues par les acteurs.

3. Il fournit un découpage fonctionnel du système d'information qui servira de base à la planification de la réalisation d'une application de base de données.

CAS 4-2 CLUB DE VOILE

(DP) À partir d'une description sommaire des activités d'un club de voile, proposer un modèle de fonctionnement de leur système d'information bien qu'il n'existe pas sur une base formelle. Procéder systématiquement en établissant d'abord les frontières d'un système ne supportant que les activités décrites dans le cas. Identifier ensuite les acteurs principaux et les acteurs auxiliaires. Dessiner le diagramme de cas d'utilisation, et rédiger tous les scénarios.

Énoncé du cas :

Le club de voile planifie en janvier de chaque année les régates qu'il compte organiser pour une saison donnée. C'est la responsabilité de son comité de régate. Une régate peut comporter plusieurs étapes, chaque étape est caractérisée par un lieu, un jour et une date prévue de départ. Il arrive cependant que la date de départ d'une étape puisse changer ou même qu'une étape soit annulée. Le comité de régate est responsable de prendre de telles décisions et d'en informer les clubs associés et les skippers inscrits.

Avant le début de la saison, le club transmet son calendrier de régates aux autres clubs de voile associés à la Fédération de voile du Québec et répond à leurs demandes d'information. Ce calendrier est transmis sur un document appelé **Calendrier des régates de la saison**. Chaque skipper est responsable d'inscrire un voilier à une régate sur la **Fiche d'inscription**. L'inscription porte sur une seule régate. Le Skipper doit joindre un chèque visé couvrant les frais d'inscription. Pour des raisons d'assurance, le comité de régate exige que tout participant à une régate soit licencié de la fédération québécoise de voile (FVQ). Il faut donc que le skipper (chef de bord effectuant l'inscription et capitaine du voilier) fournisse au comité au moment de son inscription le numéro FVQ du skipper ainsi que des membres de son équipage. Le comité s'assure qu'ils sont tous inscrits sur la **Liste des licenciés de la FVQ** pour confirmer leur inscription. Cette liste est commandée périodiquement à la Fédération par le club.

Il y a deux types de licence, les skippers (S) et les équipiers (E). Le skipper doit être le même pour toutes les étapes d'une régate mais l'équipage peut changer à chaque étape. Un même voilier peut être skippé par différents skippers licenciés d'une régate à l'autre. Les noms des membres de l'équipage doivent être communiqués au plus tard deux heures avant le départ d'une étape.

Ce sont les voiliers qui sont inscrits au classement d'une régate et non pas les skippers. Un voilier est connu par son nom, sa longueur et l'année de sa construction. Du point de vue des régates, chaque voilier appartient à une classe (il y a 3 classes en vigueur actuellement) et possède un rating. Un rating est une sorte de handicap qui permet de faire concourir des voiliers différents et de calculer le temps compensé (en multipliant le temps réel par le rating).

Le comité dresse une **Liste des participants à une régate**. Elle servira de base pour compiler un classement par étape, un classement par régate et un classement par saison. Ces documents de classement portent respectivement les noms **Classement d'une étape**, **Classement de la régate**, **Classement de la saison**. On retrouve deux types de classement applicables pour une régate : le classement en temps réel et le classement en temps compensé. Pour chaque étape et chaque régate, seul un classement basé sur les temps compensés est donné. Tous ces classements sont publiés et remis aux skippers participants.

Chaque étape doit être supervisée par un juge. Au début de la saison, après avoir pris connaissance du **Calendrier des régates de la saison**, les juges indiquent toutes les étapes pour lesquelles ils sont disponibles. Le comité affecte par la suite un juge pour chaque étape en fonction de leurs disponibilités. Entre autres fonctions, les juges doivent recueillir les réclamations. Chaque document **Réclamation** est numéroté et le juge y inscrit les parties prenantes, voiliers et skippers, ainsi qu'un résumé de l'objet du litige. Une réclamation est faite dans le contexte d'une étape et déposée auprès du juge présidant l'étape par le skipper d'un voilier. Elle peut concerner plusieurs voiliers incriminés.

Étude du cas :

Les activités décrites dans le cas mettent en lumière l'**Organisation de régates**. C'est pourquoi nous avons retenu ce nom pour le système d'information qui les supporte. Le nom montre bien la portée du système qui ne concerne aucunement les autres activités du club, telles que l'inscription des membres du club ou sa gestion financière par exemple.

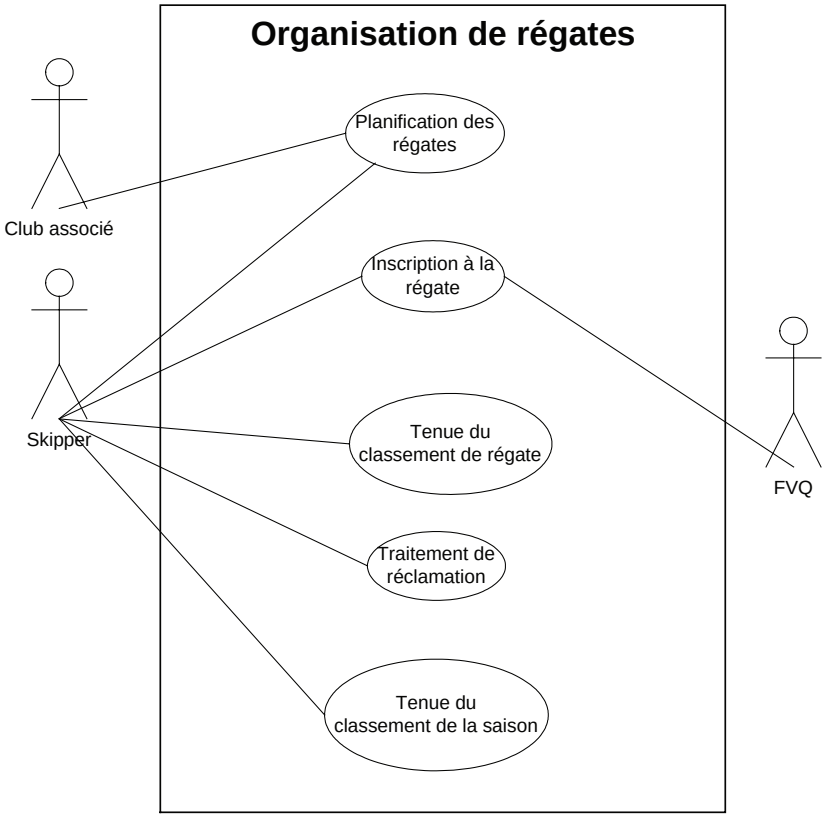
Considérant son rôle dans l'inscription d'un voilier et de son équipage à une régate, un acteur principal émerge : le skipper. Il agit nettement comme élément déclencheur de services. D'autres acteurs de moindre importance sont présents notamment à titre de prestataires de services ou bénéficiaires de services. La Fédération de voile du Québec qui fournit une **Liste des services licenciés de la FVQ** est du nombre des prestataires. Cette liste permet au club d'accepter ou de rejeter une inscription à une régate. Tout club associé est aussi un acteur car le système produit à leur intention un **Calendrier des régates de la saison**. On doit le considérer comme un bénéficiaire de services et, à ce titre, comme acteur principal.

Un grande question se pose sur le rôle joué par le juge. Agit-il à titre de ressource du système ou d'acteur externe ? Son rôle consiste à sanctionner le classement des voiliers inscrits à une régate et à recevoir les réclamations en cas de litige. Il s'agit d'activités internes au système, probablement encadrées par des protocoles et des procédures inhérentes au système d'information. Il faut prendre garde ici de confondre le rôle d'une personne dans une organisation et son appartenance à l'organisation. Le juge n'est pas obligatoirement membre du club, il pourrait donc être externe à l'organisation. Néanmoins cela n'en fait pas un acteur externe dans le contexte du système d'information qui supporte l'organisation des régates. Il en est un rouage essentiel, au même titre que les personnes qui préparent le calendrier des régates ou traitent les inscriptions des skippers et de leurs voiliers.

Nous identifions donc trois acteurs :

- le Skippeur, acteur principal, qui agit comme déclencheur de services
- le club associé, acteur principal mais bénéficiaire de services
- La Fédération de voile du Québec, acteur auxiliaire car prestataire de services.

Le diagramme de cas d'utilisation que nous proposons ne retient que ces trois acteurs. Il met en œuvre à la fois des cas d'utilisation déclenchés par les acteurs et des cas déclenchés par les ressources internes du système.



Notre discussion portera dans un premier temps sur les cas d'utilisation déclenchés par les acteurs et dans un deuxième temps sur ceux déclenchés par les ressources du système.

L'acteur Skipper déclenche deux cas d'utilisation dans deux temps différents. Seul le Skipper peut déclencher une inscription à une régate ou bien encore le traitement d'une réclamation comme le décrivent les scénarios suivants.

Scénarios	**Cas d'utilisation : *Inscription à la régate***
Principal	Le Skipper complète une **Fiche d'inscription** pour inscrire son voilier à une régate en mentionnant notamment le nom du voilier et des membres de son équipage avec pour chacun leur numéro de licence FVQ. Il y joint un chèque couvrant les frais d'inscription. Le comité de régate prend connaissance de la **Fiche d'inscription** et s'assure, en consultant la liste **Licenciés de la FVQ**, que le skipper possède une licence de type S et que chaque équipier est membre en règle de la FVQ. Si tout est en règle, le comité inscrit le voilier et son skipper dans la classe demandée sur la liste **Participants à une régate**. Sinon l'inscription est rejetée et le comité demande au Skipper de soumettre une nouvelle inscription. Lorsque le skipper apporte des changements à son équipage, la liste **Participants à une régate** est mise à jour pour refléter le changement d'équipage touchant une ou plusieurs étapes. Aucun changement n'est accepté moins de deux heures avant le début d'une étape. Pour assurer cette vérification, on doit conserver, pour chaque changement d'équipage, la date et l'heure du changement.

Scénarios	**Cas d'utilisation : *Traitement de réclamation***
Principal	À la fin d'une étape d'une régate, un skippeur soumet au juge de la régate une réclamation. Le juge contacté complète un document **Réclamation** prénuméroté et y inscrit les parties prenantes, voiliers et skippers, ainsi qu'un résumé de l'objet du litige. Le litige ne doit concerner qu'une seule étape d'une régate. Après avoir pris en délibéré la réclamation, le juge rend son verdict. Celui-ci peut comporter une sanction sous forme de pénalité de temps imposée à un ou des voiliers visés par la réclamation. La sanction peut donc mener à un changement au classement de l'étape. Son verdict est sans appel.

Le lecteur pourra noter, dans ce dernier cas d'utilisation, qu'il offre plus de détails sur le traitement d'une réclamation que l'énoncé de l'étude de cas n'en fait état. Nous voulons ainsi mettre en évidence le rôle que joue le scénario dans l'identification précise des données générées lors d'une activité de gestion et surtout l'importance de viser l'exhaustivité. Dans le scénario **Traitement de réclamation**, les données sur le verdict et la sanction imposée par le juge constituent des données dont il faudra sans doute assurer la persistance. Il incombe au modélisateur de faire ressortir dans chaque scénario toutes les activités susceptibles de générer des données persistantes.

Considérons maintenant les cas d'utilisation déclenchés par les ressources internes du système d'information. Il s'avère que tous les autres cas d'utilisation sont dans cette catégorie. Procédons à la rédaction des scénarios sur la base de la chronologie des événements qui déclenchent les scénarios.

Scénarios	**Cas d'utilisation : *Planification des régates***
Principal	Le cas débute au début d'une année de calendrier. Le comité de régate se réunit alors pour planifier les régates qui seront tenues durant la saison estivale de l'année. Une régate peut comporter plusieurs étapes. Chaque étape est caractérisée par un lieu, un jour et une heure prévue de départ. Chaque régate prévue est inscrite au document **Calendrier des régates de la saison**. Le Calendrier est transmis à tous les clubs associés avant le début de la saison. Le comité de régate transmet aussi le calendrier aux juges qui ont agi à ce titre aux cours des saisons passées en leur demandant d'indiquer leurs disponibilités pour les régates à courir lors de la prochaine saison. Chaque étape doit être supervisée par un juge. Les juges consultés fournissent toutes les étapes pour lesquelles ils sont disponibles ; le comité affecte par la suite un juge pour chaque étape en fonction de leurs disponibilités. Lorsque des changements sont apportés au **Calendrier des régates de la saison**, telle une modification à une date de départ ou l'annulation d'une étape, les clubs associés en sont informés. De plus, si des inscriptions ont été reçues pour une régate affectée par un changement, les skippers inscrits en sont aussitôt informés.

Scénarios	Cas d'utilisation : *Tenue du classement de régate*
Principal	Le cas débute lorsqu'une étape est courue. Le juge compile l'heure de départ et d'arrivée de chaque voilier puis calcule leur temps réel et leur temps compensé. Du point de vue des régates, chaque voilier appartient à une classe (il y a 3 classes en vigueur actuellement) et possède un rating. Le rating est utilisé pour calculer le temps compensé en multipliant le temps réel par le rating du voilier. Il établit alors un classement de l'étape, sur le document **Classement d'un étape,** dressé selon le temps compensé de chaque voilier participant dans une classe donnée. Si une réclamation donne lieu à une sanction pour un voilier incriminé, le verdict du juge peut entraîner un ajout au temps compensé d'un ou de plusieurs voiliers. Cet ajustement au temps compensé peut entraîner un nouveau classement de l'étape. Lorsque toutes les étapes d'une régate sont courues, il établit un classement final pour la régate en additionnant les temps compensés de chaque voilier dans chaque classe. Ce classement est publié dans le document **Classement de la régate** qui est remis aux skippers inscrits à la régate.

Scénarios	Cas d'utilisation : *Tenue du classement de la saison*
Principal	Le comité se réunit à la fin d'une saison. Il établit alors un classement des voiliers participant à toutes les régates à l'aide du document **Classement de la saison**. Le classement d'un voilier dans une régate lui confère un certain nombre de points au classement final de la saison. Une première place donne 10 points, une deuxième 7 points, etc. Le total des points accumulés au cours d'une saison pour chaque voilier inscrit aux régates du club fournit son classement pour la saison dans la classe à laquelle il appartient. Le document **Classement de la saison** est transmis à tous les skippers des voiliers inscrits au cours de la saison.

On constate que le dernier scénario fait ressortir des détails sur le calcul du classement d'une saison que l'exposé de l'étude de cas ne fournit pas. Rappelons qu'il est impératif que les scénarios fassent état des données générées par une activité de gestion et en particulier, lorsqu'il y a calcul d'une donnée numérique, le scénario doit faire état de sa méthode de calcul.

Les scénarios de cas d'utilisation ont tous été rédigés. Le modèle de fonctionnement du système d'information pour la gestion des régates répond bien aux objectifs poursuivis avec un tel modèle :

1. Le modèle de fonctionnement du système d'information constitue une représentation graphique du contexte du système et met en évidence les sources de données externes ; il permet d'inventorier les formulaires et les documents qui sont à la source des données persistantes du système ; il permet de faire voir de plus comment une donnée est générée, par exemple de quelle manière on fait le choix des juges pour une étape ou on établit le classement d'un voilier pour une saison de régate.
2. Il assure une représentation de ce que le système fait et donc de ses fonctionnalités telles que vues par les acteurs.
3. Il fournit un découpage fonctionnel du système d'information qui pourra servir de base à la planification de la réalisation d'une application de base de données.

Inventaire des documents exploités dans le système d'information et recensement des données persistantes

La production d'un modèle de fonctionnement du système d'information n'est pas un exercice sans suite. Il devrait permettre au modélisateur de recenser les données persistantes du système d'information en vue d'en faire un modèle conceptuel de données. Mais comment y arriver ?

Nous proposons la démarche suivante :

1. Parcourir tous les scénarios des cas d'utilisation et faire l'inventaire des documents mentionnés ;
2. Établir la structure de données de chaque document sous la forme d'un descriptif du document tel que préconisé au chapitre 1 ;
3. À la lecture des scénarios, relever les données générées dans le cadre d'une activité qui ne sont pas nécessairement présentes dans les documents identifiés dans les scénarios ;
4. Déterminer enfin, parmi les données présentes dans les documents ou les données qui sont générées par le système d'information mais non présentes dans les documents mentionnés, celles dont on devra assurer la persistance dans le contexte où une application de base de données serait réalisée pour automatiser les fonctions du système d'information.

S'il est vrai que l'inventaire des documents identifiés dans les scénarios représente la source première des données persistantes, les recensements des données persistantes à travers un grand nombre de documents va exiger du modélisateur une certaine vigilance pour éviter qu'il n'y ait redondance. En outre, les données dont on devra assurer la persistance ne sont pas nécessairement toutes présentes dans les documents faisant l'objet de l'inventaire. Le modélisateur devra donc être doublement vigilant. Illustrons cette problématique à l'aide de l'étude de cas 4-2.

Le cas d'utilisation **Inscription à la régate** fait ressortir qu'il existe un document appelé **Fiche d'inscription** par lequel le skipper complète son inscription et un deuxième, **Liste des participants à une régate**, qui fait état des voiliers inscrits à une régate ainsi que leur équipage. Il n'y a pas de doute que le **Nom du voilier** sera présent à la fois sur une **Fiche d'inscription** et sur la **Liste des participants à un régate**. Cela ne signifie point qu'il s'agit de deux données persistantes différentes. Cela implique cependant que le système d'information, et en conséquence l'application informatique qui le supporte, doit mémoriser le **Nom du voilier** pour chaque inscription à une régate. La **Fiche d'inscription** représente en quelque sorte la *transaction* qui a permis de recevoir l'inscription et **La Liste des participants à une régate** représente un état consolidé des inscriptions valides reçues. Par ailleurs, toujours dans le

cas 4-2, aucun document ne semble faire état de la décision du juge découlant d'une réclamation. Or il s'agit d'une donnée vitale pour le bon fonctionnement de l'organisation des régates car elle peut imposer une pénalité aux voiliers incriminés et un changement au classement de l'étape. Il y a donc lieu d'assurer la persistance de cette donnée même si, à priori, aucun document inventorié n'en fait mention.

Pour éviter le piège de la redondance, nous suggérons de procéder ainsi :

- faire l'inventaire des documents mentionnés dans les cas d'utilisation,
- produire ensuite les structures de données qu'ils comportent,
- faire un recensement des données persistantes où la redondance ne saurait être permise,
- terminer par une relecture des scénarios qui devrait permettre de recenser des données persistantes apparemment non présentes dans les documents inventoriés.

Dans le but d'illustrer la démarche, nous allons reprendre les études de cas 4-1 et 4-2 au point où nous les avions laissées et produire un recensement des données persistantes du système d'information en question.

CAS 4-3 DONNÉES PERSISTANTES DANS LE CAS TANNERIE TANBEC

Considérant la nature des documents mentionnés dans les scénarios de cas d'utilisation du projet Tannerie TANBEC, déterminer la structure de données de chaque document sous forme d'un ***Descriptif du document*** et produire un recensement des données persistantes tout en évitant la redondance mais en tenant compte de données persistantes non présentes dans les documents inventoriés.

Considérons à tour de rôle les documents dont il est fait mention dans l'énoncé du cas et a fortiori dans les deux scénarios présentés au cas 4-1.

Dossier client

Dossier client

No Client:	*1234*
Nom:	*Paul Carrier*
Adresse:	*116, Du Pont*
	Lévis, G5V 1R1
Téléphone:	*418-883-7799*

Date	***No facture***	***Débit***	***No reçu***	***Crédit***
01-02-03	*FC-3445*	*162,50 $*		
01-03-03	*FC-4824*	*230,45 $*		
15-02-03			*RC-123*	*100,00 $*
Solde à payer:		*292,95 $*		

```
Descriptif du document: Dossier client
Numéro du client
Nom du client
Adresse du client
Téléphone du client
Transaction au dossier(*)
      Date facture client
      Numéro facture client
      Total facturé
      Date reçu
      Numéro reçu
      Montant paiement
Solde à payer
```

On constate ici que l'exercice consistant à formuler la structure de données d'un document exige une vue d'ensemble des autres documents. La sous-structure appelée **Transaction au dossier** comporte, dans le cas d'un débit, des données relatives à une

facture (**Date facture client**, **Numéro facture client** et **Total facturé**) et, dans le cas d'un crédit, des données relatives à un paiement (**Date reçu**, **Numéro reçu**, **Montant paiement**). Le modélisateur aurait pu tomber dans le piège des synonymes en libellant les données de la manière suivante : Date de transaction, Montant débité et Montant crédité. Or les documents **Facture Client** et **Reçu** contiennent des données de même nature, d'où l'importance d'utiliser la même appellation pour chacune soit **Date facture client** ou **Date reçu**, **Total facturé** et **Montant paiement**.

Commande client

Commande client

No Commande: *C-123*
No Client: *1234*

Nom: *Paul Carrier*
Adresse: *116, Du Pont*
Lévis, G5V 1R1
Téléphone: *418-883-7799*

Date commande:	*12-12-02*	***Nombre peaux fournies:***	*2*
Date prévue liv.:	*12-02-03*	***Prix au cm^2:***	*0,005 $*
Date livraison:	*10-02-03*	***Surface brute:***	*1,5 m^2*
Type tannage:	*T2, castor*	***Surface tannée:***	*1,45 m^2*

Montant total *72,50 $*

Descriptif du document: *Commande client*
Numéro commande client
Numéro du client
Nom du client
Adresse du client
Téléphone du client
Date commande client
Date prévue de livraison
Date livraison
Code type de tannage
Type de peau
Nombre de peaux fournies
Prix au cm2
Surface brute
Surface tannée
Montant commande

Le document reprend bon nombre de données déjà identifiées dans le document **Dossier client**. C'est le cas pour **Numéro du client** et **Nom du client**. Aussi faut-il s'assurer d'utiliser la même appellation pour une donnée redondante et éviter les synonymes.

Le lecteur aura noté que le modélisateur a repéré deux données distinctes bien que le document fasse référence à ces données avec la même étiquette : TYPE TANNAGE. Il s'agit de **Code type de tannage** et **Type de peau**.

Tarification

Tarification

Type tannage	***Prix au cm²***	
	Peaux fournies	***Sans peaux***
T1, vache	*0,004 $*	*0,050 $*
T2, castor	*0,005 $*	*0,065 $*
T3, vison	*0,007 $*	*0,090 $*

Descriptif du document: *Tarification*
Code type de tannage
Type de peau
Prix au cm2 avec peaux
Prix au cm2 sans peaux

Commande au fournisseur

Commande au fournisseur
Tannerie TANBEC
234, Du Rocher
Lévis, G0X 3C6

No Commande:	*F-33445*	***No Fournisseur:***	*7778*
Date commande:	*12-12-02*	***Nom:***	*La Pelleterie Inc.*
No commande Client:	*C-023*	***Adresse:***	*1234-C, Du parc Lévis, G0K 1K0*
		Téléphone:	*418-833-8877*
		Télécopieur:	*418-833-7865*
Type de peau:	*Castor*		
Prix d'achat:	*200 $*		
Surface brute:	*0,5 m²*		

Descriptif du document: *Commande au fournisseur*
Numéro commande fournisseur
Date commande fournisseur
Numéro commande client
Numéro du fournisseur
Nom du fournisseur
Adresse du fournisseur
Téléphone fournisseur
Télécopieur fournisseur
Type de peau commandée
Prix achat peaux
Surface brute commandée

Ce dernier exemple illustre éloquemment que l'identification des données présentes dans un document ne consiste pas seulement à repérer les étiquettes et les libellés qu'il contient. Tannerie TANBEC est un libellé inscrit sur le document mais il ne représente pas une donnée. Il en va de même pour l'adresse de la tannerie. De plus **No commande** ne décrit pas bien la donnée qui suit cette étiquette. Pour éviter toute confusion avec une donnée présente dans le document **Commande client** qui porte la même étiquette, le modélisateur a choisi une appellation sans ambiguïté qui décrit clairement la donnée : **Numéro commande fournisseur**.

Facture client

Facture client

No Facture:	*FC-3445*	***Nom client:***	*Paul Carrier*
Date facture:	*01/02/2003*	***Adresse:***	*116, Du Pont*
No Client:	*1234*		*Lévis, G5V 1R1*
		Téléphone:	*418-883-7799*

Numéros des commandes	***Montant***
C-123	*72,50 $*
C-089	*90,00 $*
Total facturé:	*162,50 $*

```
Descriptif du document: Facture client
Numéro facture client
Date facture client
Numéro du client
Nom du client
Adresse du client
Téléphone du client
Commande(*)
      Numéro commande client
      Montant commande
Total facturé
```

La facture au client reprend plusieurs données présentes par ailleurs dans la commande du client. Il importe de conserver leur appellation. C'est le cas de **Numéro commande client** et **Montant commande**.

Reçu

Reçu

No reçu:	*R-123*
Date reçu:	*16/02/2003*
No Facture:	*FC-3445*
Nom client:	*Paul Carrier*
Montant payé:	*100,00 $*

```
Descriptif du document: Reçu
Numéro reçu
Date reçu
Numéro facture client
Nom du client
Montant paiement
```

Facture fournisseur

Facture

No Facture:	*1267*		
Date facture:	*20/12/2002*		
Nom fournisseur:	*La Pelleterie Inc.*	***Nom client:***	*Tannerie TANBEC*
Adresse:	*1234-C, Du parc*	***Adresse:***	*234, Du Rocher*
	Lévis, G0K 1K0		*Lévis, G0X 3C6*
Téléphone:	*418-833-8877*		
Télécopieur:	*418-833-7865*		
Montant facture:	*200,00 $*		

Descriptif du document: *Facture fournisseur*
Numéro facture fournisseur
Date facture fournisseur
Nom du fournisseur
Adresse du fournisseur
Téléphone fournisseur
Télécopieur fournisseur
Nom de tannerie
Adresse de tannerie
Montant facture

Il importe de bien distinguer parmi les données présentes dans les documents **Facture client** et **Facture fournisseur** celles qui possèdent des étiquettes de celles qui représentent des données distinctes. C'est notamment le cas de **Numéro facture fournisseur** et **Date facture fournisseur** qui ne doivent pas être confondues avec leur équivalent sur une facture au client : **Numéro facture client** et **Date facture client**.

Maintenant qu'a été dressé le descriptif de chaque document mentionné dans les cas d'utilisation et que chaque donnée possède une appellation explicite qui permet d'éviter les problèmes de synonymes et de la polysémie, il ne reste plus qu'à recenser les données persistantes en écartant les doublons (la même donnée présente dans plus d'un document ne doit être recensée qu'une seule fois).

Pour ce faire, il peut être utile de compiler dans une feuille de calcul le nom des données apparaissant dans les descriptifs en indiquant pour chacune dans quel document la donnée a été repérée pour la première fois, quel est son type, de quelle nature sont les contraintes d'intégrité qui s'y appliquent et s'il est pertinent ou non d'en faire une donnée persistante. La feuille de calcul présentée ci-après est le produit d'une telle compilation pour le cas TANBEC. Elle peut servir de modèle pour procéder au recensement des données persistantes en vue d'en faire un modèle conceptuel de données.

La colonne NOM DONNÉE reprend toutes les données des divers descriptifs de données sans introduire de redondance et sans la présence des noms des structures de données. Dans cette colonne, les données sont regroupées sous le nom du document où elles furent recensées la première fois. C'est le cas de **Date facture client** et **Numéro facture client** qui ont été repérées d'abord dans le document **Dossier client** mais qui apparaissent aussi dans le document **Facture client**. C'est pourquoi aucune donnée nouvelle n'est recensée dans le document **Facture client** car toutes les données présentes dans ce document ont préalablement été recensées dans **Dossier client** et **Commande client**. La présence d'un X dans la colonne P indique qu'il s'agit pour le modélisateur d'une donnée persistante. Par exemple le nom de la tannerie et son adresse ne sont pas réputées persistantes. Le **Mode** indique de quelle manière la donnée peut être reproduite. Elle peut soit être mémorisée, soit calculée à partir de données mémorisées. **Contrainte d'intégrité** limite les valeurs autorisées pour la donnée. Dans le cas d'une donnée calculée, on peut, si on le souhaite, indiquer la méthode de calcul employée dans la colonne **Règle de calcul**.

RECENSEMENT DES DONNÉES

Projet : TANBEC 16/3/2007

Nom donnée	P	Type	Mode	Contrainte d'intégrité	Règle de calcul
Dossier client					
Numéro du client	X	entier	mémorisé		
Nom du client	X	texte	mémorisé		
Adresse du client	X	texte	mémorisé		
Téléphone du client	X	texte	mémorisé		
Date facture client	X	date	mémorisé		
Numéro facture client	X	texte	mémorisé		
Total facturé	X	monnaie	mémorisé		
Date reçu	X	date	mémorisé		
Numéro reçu	X	texte	mémorisé		
Montant paiement	X	monnaie	mémorisé		
Solde à payer	X	date	calculé		
Commande client					
Numéro commande client	X	texte	mémorisé		
Date commande client	X	date	mémorisé		
Date prévue livraison	X	date	mémorisé		
Date livraison	X	date	mémorisé		
Code type de tannage	X	texte	mémorisé	T1 ou T2 ou T3	
Type de peau	X	texte	mémorisé	Vison, castor, vache, autre	

Nom donnée	P	Type	Mode	Contrainte d'intégrité	Règle de calcul
Commande client (*suite*)					
Nombre de peaux fournies	X	entier	mémorisé		
Prix au cm2	X	monnaie	mémorisé		
Surface brute	X	réel	mémorisé		
Surface tannée	X	réel	mémorisé		
Montant commande	X	monnaie	calculé		
Tarification					
Prix au cm2 avec peaux	X	monnaie	mémorisé		
Prix au cm2 sans peaux	X	monnaie	mémorisé		
Commande au fournisseur					
Numéro commande fournisseur	X	texte	mémorisé		
Date commande fournisseur	X	date	mémorisé		
Numéro du fournisseur	X	texte	mémorisé		
Nom du fournisseur	X	texte	mémorisé		
Adresse du fournisseur	X	texte	mémorisé		
Téléphone fournisseur	X	texte	mémorisé		
Télécopieur fournisseur	X	texte	mémorisé		
Prix achat peaux	X	monnaie	mémorisé		
Type de peau commandée	X	texte	mémorisé		
Surface brute commandée	X	réel	mémorisé		
Facture client					
Aucune donnée nouvelle					
Reçu					
Aucune nouvelle donnée					
Facture fournisseur					
Numéro facture fournisseur	X	entier	mémorisé		
Date facture fournisseur	X	date	mémorisé		
Nom de tannerie		texte	mémorisé		
Adresse de tannerie		texte	mémorisé		
Montant facture	X	monnaie	mémorisé		
Autres données persistantes					
Aucune autre					

L'étude de cas 4-2 ne fait état explicitement que de deux scénarios de cas d'utilisation. Ces scénarios ne font pas ressortir de données que l'on ne retrouve déjà dans l'un ou l'autre des documents dont le descriptif vient d'être établi, ce qui explique que l'on ne retrouve aucune donnée sous **Autres données persistantes**.

Comme l'illustre cette étude de cas, procéder à une compilation des données persistantes exige du modélisateur une très bonne vue d'ensemble du domaine ou du métier à modéliser. Savoir reconnaître une seule et même donnée parmi des données présentes dans plusieurs documents avec des étiquettes différentes, de manière à éviter ainsi la redondance, exige de sa part une bonne connaissance du fonctionnement du système existant ainsi qu'une bonne maîtrise de la terminologie du métier que supporte le système. D'où l'importance des scénarios de cas d'utilisation qui décrivent ce fonctionnement. Il en va de même pour la décision d'assurer ou non la persistance d'une donnée. Il doit pouvoir évaluer l'impact sur le fonctionnement du système et de l'organisation de ne pas mémoriser chaque donnée recensée. Pour ce faire, une bonne connaissance du système existant et des besoins de l'organisation est requise.

À ce stade d'avancement du projet TANBEC, le modélisateur dispose de tous les éléments pour procéder à la réalisation d'un modèle conceptuel de données en vertu des règles de modélisation décrites au chapitre 1. Le descriptif des documents permet de voir les associations un à plusieurs entre les données et propose une identification explicite de chaque donnée présente dans les documents. Le recensement des données permet d'établir la persistance de chaque donnée, ainsi que leur type et les contraintes d'intégrités applicables, tout en évitant la présence de données redondantes dans le modèle.

Nous poursuivrons l'étude du cas TANBEC un peu plus loin en proposant quelques astuces qui devraient permettre de s'acquitter efficacement de la tâche de modélisation conceptuelle en partant des descriptifs de document et du recensement des données persistantes.

CAS 4-4 DONNÉES PERSISTANTES DANS LE CAS CLUB DE VOILE

Considérant la nature des documents mentionnés dans les scénarios de cas d'utilisation du projet Club de voile, déterminer la structure de données de chaque document sous forme d'un Descriptif du document et produire un recensement des données persistantes en tenant compte des données persistantes non présentes dans les documents inventoriés.

Chaque document mentionné dans les scénarios doit faire l'objet d'un descriptif. Les documents sont illustrés ci-dessous dans l'ordre d'apparition des scénarios et sont suivis de leur descriptif respectif.

Fiche d'inscription

Fiche d'inscription
Saison 2006

Régate:	*Régate du Saguenay*
Date d'inscription:	*01/06/2006*
No inscription:	*10*
Nom du voilier:	*Urluberlu*
Année de construction:	*2002*
Rating:	*2,5*
Classe:	*Compétition*

Équipage	***Licence FVQ***	***Étapes***	***Statut***
Jean Lalonde	*1234*	*1, 2, 3, 4*	*Skipper*
Doug Atwater	*5678*	*1, 2, 3, 4*	*Équipier*
Yves Delorme	*9012*	*1, 2*	*Équipier*
Anne Delonde	*3456*	*3, 4*	*Équipier*

Frais à payer:	*250,00 $*
Montant payé:	*100,00 $*
Date paiement:	*01/06/2006*
Solde:	*150,00 $*

On notera, à l' examen du contenu du document, qu'un bloc de données se répète : les données sur les membres de l'équipage. Or, à chaque membre d'équipage est associé une ou plusieurs étapes auxquelles il devrait participer. Ce qui représente une donnée répétitive à l'intérieur d'un bloc de données répétés. Il a donc lieu de créer au descriptif une structure **Membre équipage** comportant des données répétées qui incorpore ailleurs une sous-structure **Étape** comportant la donnée répétitive **Numéro de l'étape**.

Descriptif du document: *Fiche d'inscription*
Saison de régate
Nom de régate
Date d'inscription
Numéro d'inscription
Nom du voilier
Année de construction
Rating
Classe (Compétition, Participation)
Membre équipage(*)
Prénom membre équipage
Nom membre équipage
Numéro licence FVQ
Statut membre (Skipper, Équipier)
Étape(*)
Numéro de l'étape
Frais à payer
Montant payé
Date du paiement
Solde à payer

MISE EN GARDE : Skipper et Équipier ne peuvent être considérés comme des données mais plutôt comme les valeurs possibles d'une donnée appelée **Statut membre** (sous-entendu : statut du membre de l'équipage).

Licenciés de la FVQ

Licenciés de la FVQ

No licence	*Type*	*Nom*	*Adresse*	*Date renouv.*
1234	*S*	*Jean Lalonde*	*Québec*	*12/06/2006*
3456	*S*	*Anne Delonde*	*Lévis*	*14/06/2006*
5678	*E*	*Doug Atwater*	*Westmount*	*15/06/2006*
9012	*E*	*Yves Delorme*	*Québec*	*16/06/2006*
…				

Descriptif du document: *Licenciés de la FVQ*
Numéro licence FVQ
Type de licence (S, E)
Prénom membre équipage
Nom membre équipage
Adresse membre équipage
Date renouvellement

Ce document comporte bien sûr des données répétées, mais un bloc de données ne se répète pas en regard d'une autre donnée. Il n'y a pas lieu d'en faire une structure de données. Au contraire de l'exemple précédent où une inscription identifiée par un **Numéro d'inscription** est associée à plusieurs membres d'équipage et où un membre d'équipage est associé à plusieurs étapes.

Participants à une régate

Participants à une régate
Saison 2006
Nom de la régate: *Régate du Saguenay*

No inscription	*Classe*	*Voilier*	*Skipper*	*Équipier*	*Étapes*
10	*C*	*Urluberlu*	*Jean Lalonde*	*Doug Atwater*	*1,2,3,4*
				Yves Delorme	*1,2*
				Anne Lalonde	*3,4*
11	*C*	*Engoulevent*	*Diane Proulx*	*Rock Dion*	*1,2,3,4*
				Alain Demers	*1,2,3,4*
12	*C*	*Atoka*	*Pierre Ducros*	*Paul Ducros*	*1,2,3,4*
				Sylvie Doré	*1,2,3,4*

```
Descriptif du document: Participants à une régate
Saison de régate
Nom de régate
Inscription(*)
      Numéro d'inscription
      Classe
      Nom du voilier
      Membre équipage(*)
            Prénom membre équipage
            Nom membre équipage
            Numéro licence FVQ
            Statut membre     (Skipper, Équipier)
            Étape(*)
                  Numéro de l'étape
```

Le descriptif comporte une structure complexe. Il représente cependant fidèlement les liens un à plusieurs entre les données ou les blocs de données. Une régate est associée à plusieurs inscriptions, une inscription est associée à plusieurs membres d'équipage et un membre d'équipage est associé à plusieurs étapes de la régate.

Réclamation

Réclamation
Saison 2006

Numéro: 0102
Nom de la régate: *Régate du Saguenay*
Étape objet du litige*: 2*
Juge: *Rodolphe Ouellet*

No inscription	*Classe*	*Voilier*	*Skipper*
10	*C*	*Urluberlu*	*Jean Lalonde*

Voiliers incriminés: *Atoka, Engoulevent*
Objet du litige: *Les voiliers Atoka et Engoulevent avaient passé la ligne de départ au moment où le coup de départ a été donné*

```
Descriptif du document: Réclamation
Saison de régate
Numéro de réclamation
Nom de régate
Numéro de l'étape
Nom du juge
Prénom du juge
Numéro d'inscription
Classe
Nom du voilier
Nom membre équipage
Prénom membre équipage
Voilier incriminé(*)
      Nom du voilier
Objet du litige
```

Le descriptif du document **Réclamation** montre bien les liens un à un existant entre une réclamation et toutes les données du document sauf une : le voilier incriminé. Une réclamation peut en effet mettre en cause plusieurs voiliers incriminés. La donnée réelle que véhicule la structure **Voilier incriminé** est le nom d'un voilier. Ce qui explique que **Nom du voilier** apparaît deux fois dans le descriptif : une première fois pour l'identification du voilier qui fait l'objet de la réclamation et une deuxième fois, dans une structure répétitive, pour faire état des voiliers incriminés. La structure **Voilier incriminé** est doublement justifiée. D'abord pour mettre en évidence la répétition et par ailleurs pour faire une nette distinction entre les deux occurrences de la donnée **Nom du voilier**.

Ce descriptif montre bien les liens existant entre une réclamation et un voilier. Ces liens devront être traduits dans le modèle conceptuel de données. Les entités **Réclamation** et **Voilier** devront être forcément liés par deux associations : **Voilier réclamant** et **Voilier incriminé**.

Calendrier des régates de la saison

Calendrier des régates de la saison
Saison 2006

Régate: *Régate du Saguenay*
Nombre d'étapes: *4*
Date limite d'inscription: *01/05/2006*
Nombre de places: *30*

Étape	*Départ*	*Arrivée*	*Date départ*	*Heure départ*
1	*Québec*	*Tadoussac*	*12/06/2006*	*9h30*
2	*Tadoussac*	*Baie des Ha-Ha*	*14/06/2006*	*9h00*
3	*Baie des Ha-Ha*	*Cap à l'aigle*	*15/06/2006*	*9h30*
4	*Cap à l'aigle*	*Québec*	*16/06/2006*	*9h00*

Régate: *Tour de l'île*
Nombre d'étapes: *1*
Date limite d'inscription: *01/05/2006*
Nombre de places: *50*

Étape	*Départ*	*Arrivée*	*Date départ*	*Heure départ*
1	*Québec*	*Québec*	*15/07/2006*	*9h30*

Descriptif du document: *Calendrier des régates de la saison*
Saison de régate
Régate(*)
- *Nom de régate*
- *Nombre d'étages*
- *Date limite d'inscription*
- *Nombre de places*
- ***Étape(*)***
  - *Numéro de l'étape*
  - *Point de départ*
  - *Point d'arrivée*
  - *Date de départ*
  - *Heure de départ*

La saison de la régate est une donnée à laquelle on associe plusieurs régates caractérisées notamment par **Nom de régate** et **Nombre d'étapes**. De plus, chaque régate est associée à plusieurs étapes avec leurs données propres.

Classement d'une étape

Classement d'une étape
Saison 2006
Nom de la régate: *Régate du Saguenay*
Numéro de l'étape*: 1*

Classe: *Compétition*

Classement	***Voilier***	***Temps réel***	***Temps compensé***
1	*Atoka*	*2h35*	*2h05*
2	*Engoulevent*	*2h40*	*2h10*
3	*Urluberlu*	*2h43*	*2h12*

…

Classe: *Participation*

Classement	***Voilier***	***Temps réel***	***Temps compensé***
1	*Croix du sud*	*2h50*	*2h35*
2	*Engoulevent*	*2h45*	*2h42*

```
Descriptif du document: Classement d'une étape
Saison de régate
Nom de régate
Numéro de l'étape
Classement(*)
      Classe
      Résultat(*)
      Rang étape
      Nom du voilier
      Temps réel
      Temps compensé
```

Ce descriptif montre bien qu'une étape est liée à plusieurs classes et chaque classe comporte plusieurs résultats regroupant chacun : **Rang**, **Nom du voilier**, **Temps réel** et **Temps compensé**.

Les deux derniers documents illustrés ci-après, **Classement de la régate** et **Classement de la saison**, possèdent des structures similaires mais introduisent de nouvelles données, soit respectivement **Temps compensé total** et **Total des points**.

Classement de la régate

Classement de la régate
Saison 2006
Nom de la régate: *Régate du Saguenay*

Classe: *Compétition*

Classement	***Voilier***	***Temps compensé total***
1	*Atoka*	*15h35*
2	*Engoulevent*	*16h40*
3	*Urluberlu*	*17h43*

…

Classe: *Participation*

Classement	***Voilier***	***Temps compensé total***
1	*Croix du sud*	*20h50*
2	*Engoulevent*	*21h45*

…

```
Descriptif du document: Classement de la régate
Saison de régate
Nom de régate
Classement(*)
      Classe
      Résultat(*)
            Rang régate
            Nom du voilier
            Temps compensé total
```

Classement de la saison

Classement de la saison*: 2006*

Classe: *Compétition*

Classement	***Voilier***	***Total des points***
1	*Atoka*	*97*
2	*Engoulevent*	*89*
3	*Urluberlu*	*65*

…

Classe: *Participation*

Classement	***Voilier***	***Total des points***
1	*Croix du sud*	*45*
2	*Engoulevent*	*32*

…

Descriptif du document: *Classement de la saison*
Saison de régate
Classement(*)
Classe
Résultat(*)
Rang saison
Nom du voilier
Total des points

RECENSEMENT DES DONNÉES

Projet : Club de voile Date : 16/3/2007

Nom donnée	P	Type	Mode	Contrainte d'intégrité	Règle de calcul
Fiche d'inscription					
Saison de régate	X	entier	mémorisé		
Nom de régate	X	texte	mémorisé		
Date d'inscription	X	texte	mémorisé		
Numéro d'inscription	X	entier	mémorisé		
Nom du voilier	X	texte	mémorisé		
Année de construction	X	entier	mémorisé		
Rating	X	réel	mémorisé		
Classe	X	texte	mémorisé	Compétition ou Participation ou Autre	
Prénom membre équipage	X	texte	mémorisé		
Nom membre équipage	X	texte	mémorisé		
Numéro licence FVQ	X	entier	mémorisé		
Statut membre	X	texte	mémorisé	Skipper ou Équipier	
Numéro de l'étape	X	entier	mémorisé		
Frais à payer	X	monnaie	mémorisé		
Montant payé	X	monnaie	mémorisé		
Date du paiement	X	date	mémorisé		
Solde à payer	X	monnaie	calculé		
Licenciés de la FVQ					
Type de licence	X	texte	mémorisé	S ou E	
Adresse membre équipage	X	date	mémorisé		
Date renouvellement	X	date	mémorisé		

Nom donnée	P	Type	Mode	Contrainte d'intégrité	Règle de calcul
Participants à une régate					
Réclamation					
Numéro de réclamation	X	entier	mémorisé		
Nom du juge	X	texte	mémorisé		
Prénom du juge	X	texte	mémorisé		
Objet du litige	X	texte	mémorisé		
Calendrier des régates de la saison					
Nombre d'étapes	X	entier	mémorisé		
Date limite d'inscription	X	date	mémorisé		
Nombre de places	X	entier	mémorisé		
Point de départ	X	texte	mémorisé		
Point d'arrivée	X	texte	mémorisé		
Date de départ	X	date	mémorisé		
Heure de départ	X	date	mémorisé		
Classement d'une étape					
Rang étape	X	entier	mémorisé		
Temps réel	X	réel	calculé		
Temps compensé	X	réel	calculé		
Classement de la régate					
Rang régate	X	entier	mémorisé		
Temps compensé total	X	réel	calculé		
Classement de la saison					
Rang saison	X	entier	mémorisé		
Total des points	X	réel	calculé		
Autres données persistantes					
Statut inscription	X	texte	mémorisé	Acceptée ou Rejetée	
Date changement d'équipage	X	date	mémorisé		
Heure changement d'équipage	X	date	mémorisé		
Heure départ du voilier	X	date	mémorisé		
Heure arrivée du voilier	X	date	mémorisé		
Pénalité au voilier	X	réel	mémorisé		
Disponibilité du juge	X		association		
Affectation du juge	X		association		

Après avoir complété le recensement des données à partir des descriptifs de document, le modélisateur a relu les scénarios de cas d'utilisation pour recenser les données persistantes dont les descriptifs ne font pas état.

En consultant le scénario de **Inscription à la régate**, on constate que le traitement d'une inscription peut donner lieu à une acceptation ou à un rejet. Il a jugé vital de conserver toutes les demandes d'inscription et de les distinguer par la donnée **Statut inscription**. Toujours dans le même scénario, on note que pour traiter un changement à l'équipage, il est nécessaire de conserver la date et l'heure du changement, d'où les nouvelles données persistantes, **Date changement d'équipage** et **Heure changement d'équipage**, qui n'avaient jusqu'alors pas été recensées sur l'unique base des descriptifs.

Le scénario **Tenue du classement de régate** indique clairement que le juge compile l'heure de départ et l'heure d'arrivée de chaque voilier participant pour calculer le temps réel et le temps compensé du voilier. Il apparaît donc vital d'en faire deux données persistantes.

Le traitement d'un réclamation donne lieu à une décision qui prend le cas échéant la forme d'une pénalité de temps imposée à l'un ou l'autre des voiliers incriminés. D'où la présence de la donnée **Pénalité au voilier** au recensement final.

Le scénario du cas **Planification des régates** fait état d'un processus où les juges manifestent leur disponibilité pour officier à des étapes. Le club affecte ensuite un juge à chaque étape selon les disponibilités. Le cas ne fait pas état de données spécifiques assurant la persistance de telles données mais le modélisateur les ajoute au recensement car elles pourraient être modélisées par des associations dans le modèle conceptuel de données.

Le cas 4-4 montre bien que le recensement complet des données persistantes exige de procéder à toutes les étapes de la démarche préconisée, la dernière incluse :

- faire l'inventaire des documents mentionnés dans les cas d'utilisation,
- produire ensuite un descriptif pour chacun,
- faire un recensement des données persistantes où la redondance ne saurait être permise,
- terminer par une relecture des scénarios qui devrait permettre de recenser des données persistantes apparemment non présentes dans les documents inventoriés.

La dernière étape met en évidence l'importance des scénarios, notamment pour formuler les règles de gestion du domaine à l'étude mais aussi pour en déduire les données persistantes requises pour pouvoir les appliquer.

Comme nous l'indiquions au début de ce chapitre, avant d'entreprendre la conception et la réalisation d'une application de base de données, le modélisateur peut compter sur deux approches pour établir les besoins en matière de données. L'alternative est la suivante : débuter par la production d'un modèle de fonctionnement du système d'information ou débuter par la production d'un modèle de fonctionnement de l'application. Quelle que soit l'approche adoptée au début du processus de définition des besoins, le résultat sera le même : un recensement des données persistantes et leur représentation sous forme d'un modèle conceptuel de données.

La prochaine section traite de cette deuxième approche et montre comment on arrive, en vertu de cette approche, à recenser les données persistantes.

Une approche ascendante : le modèle de fonctionnement de l'application

L'approche proposée dans cette section est appropriée dans la situation où l'application à réaliser ne peut s'appuyer sur un système d'information formel ou informel. Elle peut aussi être utilisée si on désire faire abstraction d'un système d'information existant. Cette décision peut se justifier en présence d'un système d'information bien réel mais devenu inapproprié, dépassé ou comportant de trop nombreuses lacunes. Il serait alors inutile de reproduire dans une application informatique les mêmes lacunes ou les mêmes carences, notamment au plan des données.

Cette approche s'appuie sur une prémisse : on peut arriver à faire une étude de besoins en matière de données pour une application de base de données en débutant par le dessin d'esquisses de panoramas d'écran pour la

nouvelle application. Les esquisses de panoramas, à partir desquelles on pourra visualiser les données persistantes et établir des règles applicables à leur gestion, seront élaborées avec le concours des utilisateurs potentiels de l'application. L'ensemble des panoramas représente un prototype *visuel* de l'application à réaliser.

Prototype visuel

Un ensemble de dessins de panoramas qui représentent chacun un instantané d'un certain nombre d'écrans de l'application à concevoir et à réaliser techniquement. Le prototype visuel est le pendant du story-board utilisé par les artisans du cinéma pour illustrer le contenu et l'enchaînement des plans pour un dessin animé ou un film. Il permet de recenser les données qui seront exploitées par l'application et notamment les données persistantes (*Visual prototype*).

Le prototype visuel ne saurait exister seul. Il sera complété par un diagramme de cas d'utilisation de l'application. Il s'agit bien du même formalisme que nous avons exploité plus tôt pour produire un modèle de fonctionnement d'un système d'information. Rappelons qu'un diagramme de cas d'utilisation permet de modéliser les fonctionnalités d'un *système* ainsi que les interactions des acteurs avec les fonctionnalités de ce *système* sous la forme de scénarios. Cette définition est applicable à tout type de système, que ce soit un système d'information ou un système informatisé.

Nous présentons ci-dessous les trois objectifs poursuivis par la réalisation d'un modèle de fonctionnement d'une application de base de données. Ils correspondent à peu de chose près à ceux d'un modèle de fonctionnement d'un système d'information. Pour mettre en évidence les différences, le texte souligné fait apparaître ce qui distingue un modèle de fonctionnement d'une application d'un modèle de fonctionnement d'un système d'information.

Les objectifs de la démarche de représentation du fonctionnement d'une application de base de données, peuvent se résumer ainsi :

1. Le modèle de fonctionnement de l'application doit constituer une représentation graphique de l'application à réaliser telle que vue par les utilisateurs ;
   - il permet d'illustrer un ensemble de panoramas d'écran, appelé le prototype visuel, qui constitue une vue concrète des formulaires électroniques devant assurer la saisie et la consultation des données persistantes de l'application ;

- il permet d'identifier les utilisateurs de l'application et de mettre en valeur les cas d'utilisation que chaque utilisateur est susceptible d'exploiter, tant pour la saisie des données que pour la génération de nouvelles données dont on souhaite assurer la persistance par le biais d'une base de données ;
- il est à l'origine de la réalisation d'un modèle conceptuel de données, produit après le recensement des données persistantes présentes dans les formulaires électroniques ou dérivé des scénarios de cas d'utilisation.

2. Les cas d'utilisation doivent assurer une représentation de ce que fait l'application et donc de ses fonctionnalités telles que vues par les acteurs (utilisateurs ou autres systèmes externes à l'application) ;
   - ils décrivent le comportement de l'application en regard de fonctionnalités logiquement regroupées : par exemple le traitement informatique des admissions dans le collège, le traitement informatique des inscriptions à chaque session, le traitement informatique pour l'émission des diplômes ;
   - ils sont axés sur les fonctions de l'application et sur les données qu'elles consomment ou qu'elles génèrent.
3. Les cas d'utilisation doivent permettre d'assurer le découpage fonctionnel de l'application de manière à permettre la planification du processus d'analyse, de conception et de réalisation ce cette application ;
   - le découpage fonctionnel devrait faciliter la planification des phases successives de réalisation de l'application ;
   - à l'aide d'un tel découpage, la planification consistera à produire un ordonnancement chronologique des cas d'utilisation (donc des fonctionnalités) à mettre en œuvre dans l'application de base de données.

Difficultés rencontrées dans la réalisation du modèle de fonctionnement d'une application

La production du modèle de fonctionnement d'une application confronte le modélisateur à deux difficultés :

1. esquisser les panoramas d'écran avec les utilisateurs ;
2. découvrir les cas d'utilisation en tentant de regrouper les fonctionnalités du système par blocs, chaque bloc de fonctionnalités logiquement apparentées représentant une fonctionnalité de haut niveau, et les rattacher ensuite aux acteurs.

Ces difficultés ne sont pas de même nature ni de même envergure que celles concernant la modélisation du fonctionnement d'un système d'information. Par exemple, le choix des acteurs pose habituellement peu de difficulté. En effet, les acteurs principaux sont les utilisateurs directs de l'application, les acteurs auxiliaires sont des systèmes externes qui procurent des services à l'application. Par ailleurs, on n'a pas à se soucier de faire la distinction entre un acteur et une personne qui agit comme ressource interne au système car une application informatique ne dispose que de ressources techniques : le matériel et le logiciel.

Le dessin des panoramas doit être réalisé dans un premier temps. Le modélisateur s'entoure alors d'un nombre restreint d'utilisateurs directs ou indirects de l'application ou de personnes bien au fait des besoins de ces utilisateurs et sur lesquelles il peut compter. Le choix des participants à l'exercice de dessin des panoramas est une difficulté en soi. Il est préférable de débuter avec un nombre restreint de participants qui ont une connaissance globale des besoins des utilisateurs, quitte à s'adjoindre par la suite des personnes qui ont une connaissance plus fine des besoins sur des aspects pointus des fonctionnalités de l'application.

Ce n'est qu'après avoir réalisé plusieurs dessins de panoramas que le modélisateur pourra s'attaquer à l'identification des cas d'utilisation et à la rédaction des scénarios de cas d'utilisation, qui devront décrire comment l'utilisateur interagit avec l'application par le biais des panoramas d'écran.

La démarche de réalisation du modèle de fonctionnement d'une application de base de données est illustrée par la prochaine étude de cas. Elle concerne la modélisation d'une application informatique qui viendra supporter la gestion des activités d'un club offrant à ses membres des séances d'entraînement ainsi que la participation à des compétitions dans diverses disciplines olympiques. On aura le loisir, à la suite de cette étude de cas, de dresser un parallèle entre la démarche de réalisation d'un modèle de fonctionnement d'une application et la démarche de réalisation d'un modèle de fonctionnement d'un système d'information tel que décrit à la section précédente. Néanmoins, quelle que soit la démarche suivie, le résultat premier reste le même : recenser les données persistantes pour produire une modèle conceptuel de données.

CAS 4-5 DONNÉES PERSISTANTES DANS L'APPLICATION CLUB D'ATHLÉTISME

Considérant la mise en situation du projet d'informatisation abordée ici, ainsi que la nature des documents électroniques représentant les panoramas d'écran de l'application souhaitée, produire un diagramme de cas d'utilisation avec tous les scénarios de cas d'utilisation du projet. Déterminer la structure de données de chaque document électronique sous forme d'un **Descriptif du document** et produire un recensement des données persistantes en tenant compte de données qui ne seraient pas présentes dans les documents inventoriés.

Portée globale de l'application de base de données : L'application doit permettre de gérer informatiquement les dossiers des membres d'un club d'athlétisme, notamment leur inscription, les données sur les performances des athlètes et les responsabilités des membres relativement à diverses activités. Il doit offrir un support à l'organisation et au suivi des activités du club. Le club peut accueillir des athlètes dans une ou plusieurs disciplines olympiques.

Contexte de son utilisation :

1. Le membre peut être un membre athlète, c'est-à-dire qu'il peut participer à des compétitions internes organisées par le club ou organisées à l'extérieur par d'autres clubs.
2. Le membre peut être un bénévole, auquel cas il n'a pas à payer les frais annuels requis d'un membre athlète. Mais alors il peut être appelé alors à participer à l'organisation des activités de compétition, aux activités de financement et à la logistique requise pour le transport, l'hébergement et l'encadrement des athlètes du club, tout comme le membre athlète. Pour jouer ce rôle, le membre doit faire partie d'un comité d'organisation.
3. L'inscription annuelle confère automatiquement à la personne sa qualité de membre et son statut (bénévole ou athlète).
4. Toute activité, financement ou compétition, implique des bénévoles et des athlètes réunis en comité. Toute activité relève d'un comité d'organisation, y compris les compétitions tenues à l'extérieur. Dans ce dernier cas, le comité devra de plus organiser le transport et le logement des athlètes ainsi que des membres accompagnateurs. Le transport peut être assuré par les bénévoles (moyennant remboursement pour les frais) ou par location de véhicules. Le logement se fait à l'hôtel ou chez des individus offrant le gîte aux membres visiteurs.
5. Toute activité comporte des participants et des organisateurs. Dans le cas d'une activité de compétition tenue à l'extérieur, le club doit procéder à une sélection des athlètes qui y participeront. Un comité d'organisation local aura la responsabilité de cette sélection. Le comité pourrait aussi prendre en charge la logistique de participation à la compétition extérieure.

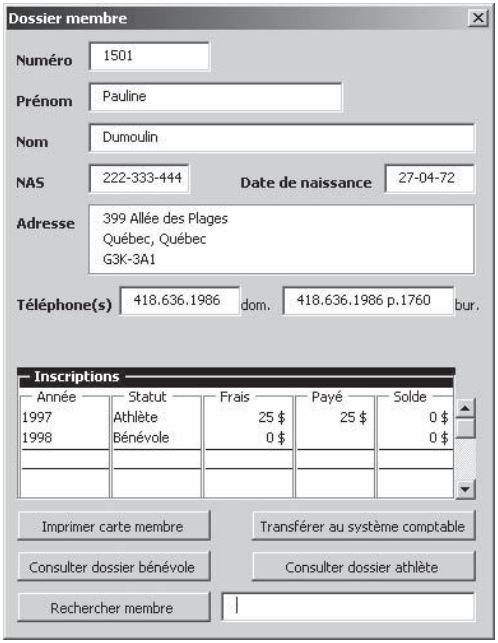
6. Le système devrait permettre de savoir, par exemple, quels sont les membres qui ont participé à telle activité et à quel titre. S'il s'agit d'une compétition, le classement de l'athlète doit être disponible, de même que son résultat dans une spécialité.
7. Les revenus et dépenses (membership, financement, compétition) sont pris en compte par le système mais transférés à un système informatisé de comptabilité. Aucune gestion comptable ou financière n'est effectuée par l'application. Le solde des comptes à recevoir des membres athlètes doit être disponible ainsi que les frais d'inscription payés au cours des ans. Les revenus et dépenses, quelles que soient les activités, devraient être disponibles même après le transfert au système comptable.
8. Les utilisateurs du système peuvent être la direction du club, un entraîneur, un directeur d'un comité d'organisation ou le personnel administratif du club.

Production d'un prototype visuel

Le modélisateur débute son travail en élaborant avec la direction du club et la personne en charge de l'administration du club des esquisses de panoramas d'écran. Il convient de dessiner en premier lieu le panorama utilisé pour assurer l'inscription des membres. Il existe de nombreux outils permettant de réaliser des documents électroniques ou des formulaires agissant comme interface utilisateur d'une application. Il suffit de mentionner Adobe Acrobat, Microsoft Visio ou le module Visual Basic de la suite Office de Microsoft. Ces outils permettent de dessiner des panoramas comportant une ou plusieurs fenêtres dotées de contrôles : boutons de commande, zones de texte pour la saisie, barres de défilement, étiquettes, formes géométriques, onglets, sélecteurs, etc.

Le premier panorama représente le document électronique utilisé pour la saisie et la consultation des données sur les membres ainsi que pour l'exécution de certaines opérations : impression de la carte de membre ou ouverture d'une deuxième fenêtre faisant voir les données spécifiques à un membre athlète par exemple.

Après avoir convenu ensemble de la pertinence de l'esquisse et de son contenu, le modélisateur dresse ensuite un descriptif du document électronique. Le résultat de son travail est décrit ci-dessous.



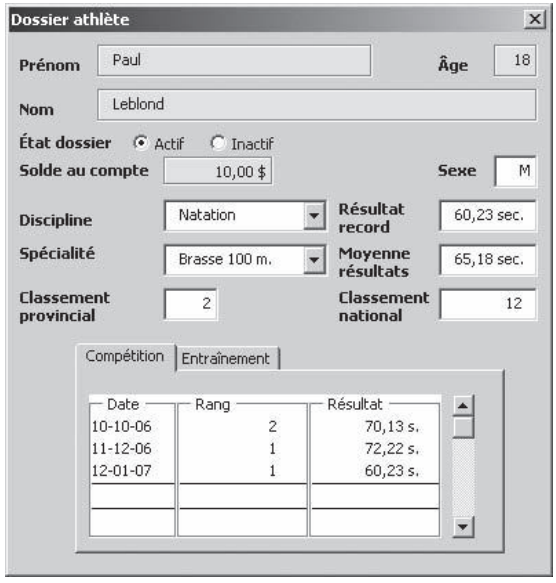
Descriptif du document: *Dossier membre*
Numéro membre
Prénom membre
Nom membre
NAS membre
Date de naissance membre
Adresse membre
Téléphone domicile
Téléphone bureau
Inscription(*)
    Année inscription
    Statut membre *(Athlète, bénévole)*
    Frais inscription
    Montant payé
    Solde

Comme pour un document papier, le modélisateur a recensé les données et les structures de données identifiables sur le document, sans égard aux contrôles qui peuvent s'y retrouver. La présence du bouton **Imprimer carte membre** ne constitue pas une donnée. Néanmoins, dans le scénario de cas d'utilisation qui décrit l'interaction d'un acteur avec

l'application qui affiche une telle fenêtre, il pourrait être écrit : « En appuyant sur le bouton Imprimer carte membre, la carte de membre sera produite et le système doit alors conserver une donnée spécifiant que la carte du membre a été produite pour sa dernière inscription ».

Dans ce cas, une donnée booléenne appelée **Carte membre imprimée ?** devrait être recensée mais seulement après la rédaction des scénarios de tous les cas d'utilisation qui sera suivie d'une révision en vue d'y recenser d'autres données persistantes. Nous n'en sommes pas encore là. Pour l'instant le modélisateur ne doit s'en tenir qu'aux données explicitement présentes dans le document électronique.

Un deuxième document a été ensuite dessiné. Il représente cette fois la fenêtre qui devrait apparaître à l'écran lorsque le bouton **Consulter dossier athlète** est actionné dans la fenêtre **Dossier membre**. Il comporte des données déjà recensées dans le document précédent : **Prénom membre**, **Nom membre** et **Solde**. Par ailleurs, la dynamique du document électronique doit être ici précisée pour comprendre comment le modélisateur en est arrivé au descriptif donné plus bas.

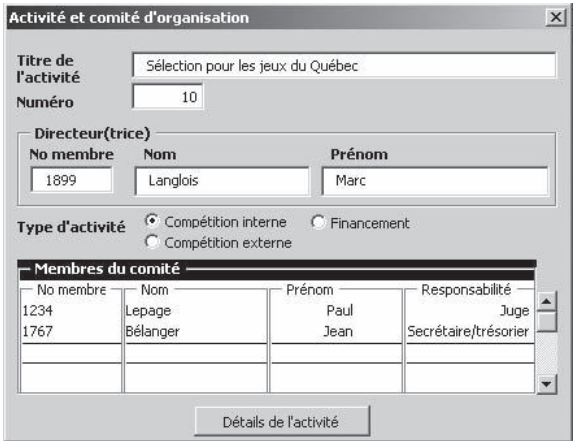


Les contrôles **Discipline** et **Spécialité** sont du type « zone déroulante », ce qui signifie que l'utilisateur peut techniquement consulter toutes les disciplines pour lesquelles l'athlète a cumulé des résultats, en sélectionner une pour voir les spécialités de l'athlète dans cette discipline à travers la deuxième zone déroulante et dès que le choix d'une spécialité est fait,

les résultats dans cette spécialité s'affichent. L'athlète est donc associé à plusieurs disciplines, chaque discipline est associée à plusieurs spécialités et une spécialité est associée à plusieurs résultats.

```
Descriptif du document: Dossier athlète
Prénom membre
Nom membre
Âge membre
État du dossier
Solde
Sexe membre
Discipline(*)
      Nom discipline
      Spécialité(*)
            Nom spécialité
            Résultat record
            Unité mesure record (sec., mètre, autre)
            Résultat moyen
            Unité mesure moyen (sec., mètre, autre)
            Classement national
            Classement provincial
            Résultat athlète(*)
                  Date résultat
                  Rang
                  Résultat
                  Unité mesure résultat(sec., mètre, …)
                  Type résultat     (entraînement,
                                    compétition)
```

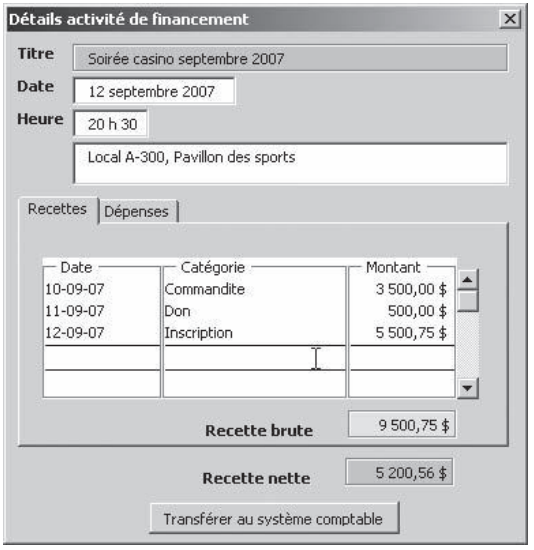
Le document électronique suivant sera utilisé pour la saisie et la consultation des données sur les activités organisées par le club ou les activités auxquelles le club participe.



Il donne lieu à un descriptif où apparaît une structure de données non répétitive, **Directeur**, qui permet de faire état du rôle joué par un membre à titre de directeur de l'activité.

```
Descriptif du document: Activité et comité
d'organisation
Numéro activité
Titre activité
Directeur
      Numéro membre
      Nom membre
      Prénom membre
Type d'activité (Comp. int., comp. ext., fin.)
Composition comité(*)
      Numéro membre
      Nom membre
      Prénom membre
      Responsabilité (Juge, secrétaire/t., autre)
```

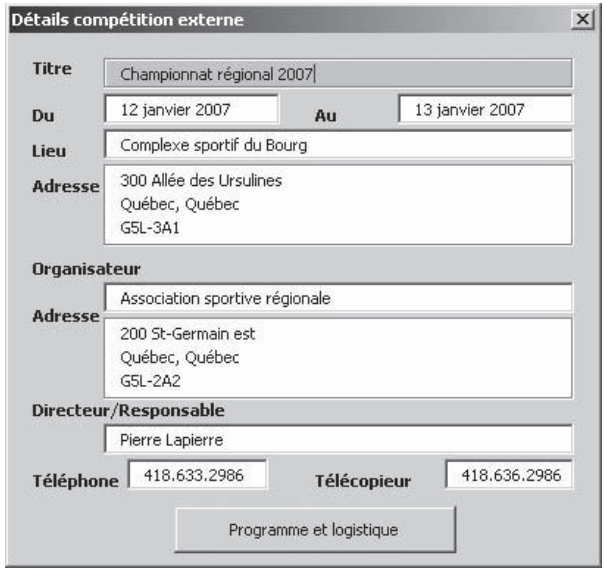
On a procédé ensuite au dessin du document électronique permettant la saisie et la consultation de données spécifiques aux activités de financement.



Son descriptif fait état à la fois des données présentes sous l'onglet Recettes (visible dans l'illustration) et sous l'onglet Recettes (visible sur activation).

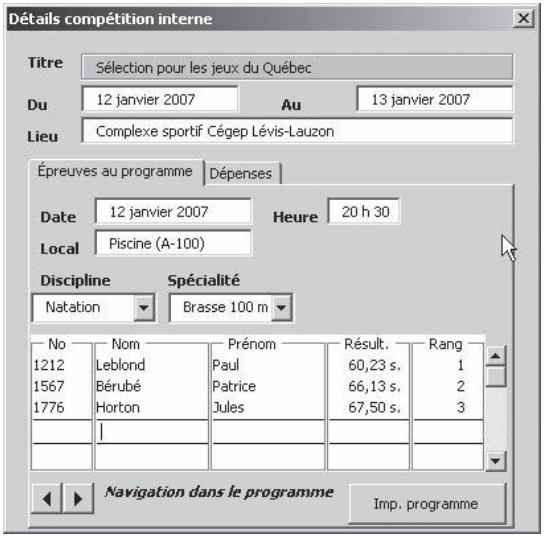
```
Descriptif du document: Détails activité de
financement
Titre activité
Date activité
Heure activité
Adresse activité
Recette(*)
      Date recette
      Catégorie recette (Commandite, dons, …)
      Montant recette
Recette brute
Dépense(*)
      Date dépense
      Catégorie dépense
      Montant dépense
Dépenses totales
Recette nette
```

La saisie et la consultation des données spécifiques à une compétition externe donnent lieu à un document électronique nouveau avec son propre descriptif.



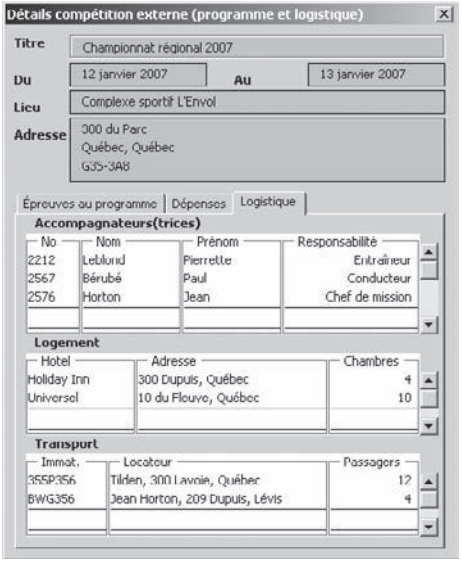
Descriptif du document: *Détails compétition externe*
Titre activité
Date début compétition
Date fin compétition
Lieu compétition
Adresse activité
Nom organisateur
Adresse organisateur
Nom responsable
Prénom responsable
Téléphone responsable
Télécopieur responsable

Les compétitions internes exigent la mise en œuvre d'un programme des épreuves et la comptabilisation des résultats et des dépenses. Le document électronique qui suit a été dessiné à cette fin. Notons qu'une épreuve ne concerne qu'une seule spécialité dans une discipline donnée.



Descriptif du document: *Détails compétition interne*
Titre activité
Date début compétition
Date fin compétition
Lieu compétition
Épreuve(*)
Date épreuve
Heure épreuve
Local épreuve
Nom discipline
Nom spécialité
Participant(*)
Numéro membre
Nom membre
Prénom membre
Résultat
Unité mesure
Rang
Dépense(*)
Date dépense
Catégorie dépense
Montant dépense

Le document électronique **Détails compétition externe** étudié plus haut comporte un bouton **Programme et logistique** qui active à l'écran une deuxième fenêtre affichant le document électronique **Détails compétition externe (Programme et logistique)**.



Les onglets **Épreuves au programme** et **Dépenses** comportent les mêmes données que les onglets correspondants dans le document **Détails compétition interne**. Les données sous l'onglet **Logistique** sont pour un bon nombre nouvelles.

```
Descriptif du document: Détails compétition
externe (programme et logistique)
Titre activité
Date début compétition
Date fin compétition
Lieu compétition
Adresse activité
Épreuve(*)
      Date épreuve
      Heure épreuve
      Local épreuve
      Nom discipline
      Nom spécialité
      Participant(*)
            Numéro membre
            Nom membre
            Prénom membre
            Résultat
            Unité mesure
            Rang
Dépense(*)
      Date dépense
      Catégorie dépense
      Montant dépense
Accompagnateur(*)
      Numéro membre
      Nom membre
      Prénom membre
      Responsabilité (Juge, secrétaire/t., autre)
Logement(*)
      Nom établissement
      Adresse établissement
      Nombre chambres
Transport(*)
      Immatriculation véhicule
      Nom propriétaire
      Adresse propriétaire
      Nombre de passagers
```

À ce stade d'avancement de la production du prototype visuel, le modélisateur et les principaux utilisateurs considèrent avoir couvert l'essentiel des fonctionnalités de l'application à réaliser. Il devront s'attaquer au deuxième volet du modèle de fonctionnement de l'application, le diagramme de cas d'utilisation et les scénarios sous-jacents.

Réalisation des cas d'utilisation

Le travail débute par une identification des utilisateurs de l'application, qui agiront comme acteurs principaux. Les autres acteurs, les acteurs auxiliaires, sont essentiellement les systèmes prestataires de services pour l'application.

Rappelons en quoi consiste les notions d'acteur principal et d'acteur secondaire en reprenant les descriptions que nous présentions plus tôt.

« Un acteur principal est un DÉCLENCHEUR DE SERVICE pour le système à modéliser ou un BÉNÉFICIAIRE DE SERVICE. Il peut s'agir d'une personne, un groupe de personnes, d'une organisation ou d'un autre système. Ce qui intéresse le modélisateur c'est la nature des données transmises par l'acteur pour déclencher le service requis ainsi que les échanges de données subséquents qui vont assurer la prestation du service à cet acteur principal. »

Dans le cas d'une application informatique, ce ne saurait être que des utilisateurs de l'application.

« Un acteur auxiliaire est un prestataire externe de services auquel le système peut faire appel pour répondre à un service déclenché généralement par un acteur principal. Son fonctionnement interne est sans intérêt pour le modélisateur. »

Pour la modélisation du fonctionnement d'une application informatique ce ne saurait être que d'autres systèmes, d'autres applications ou des logiciels qui seront couplés à l'application par un mécanisme d'échange de données.

Le cas fait explicitement référence aux utilisateurs suivants : la direction du club, un entraîneur, un directeur d'un comité d'organisation ou le personnel administratif du club. Il seront considérés comme les acteurs principaux par le modélisateur. Il n'existe qu'un système informatisé qui soit couplé à l'application. Il vise à assurer la gestion comptable des revenus et dépenses du club. Nous l'identifions sous l'appellation **Système informatisé de comptabilité**. Il s'agit pour le modélisateur d'un acteur auxiliaire.

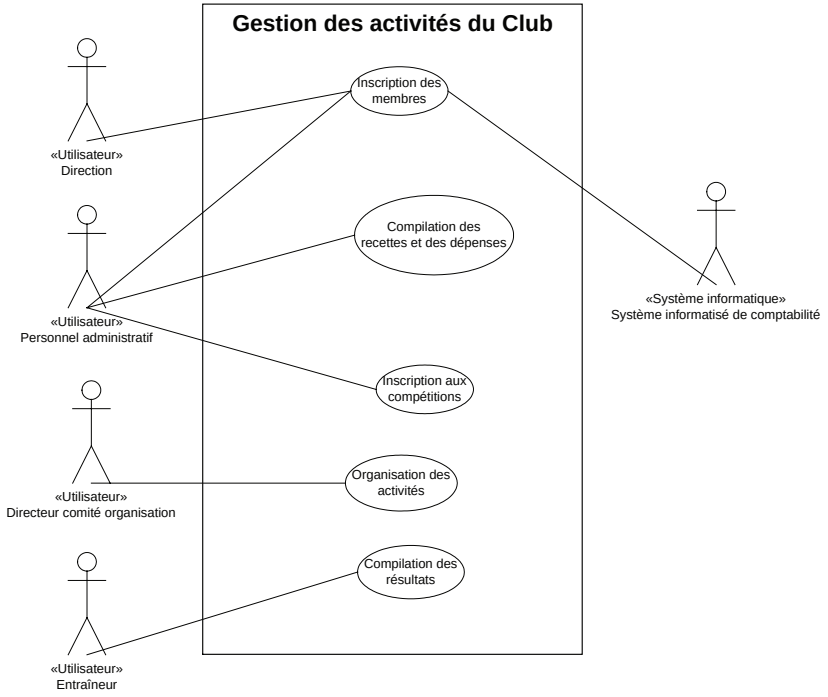
L'étape suivante consiste à découper les fonctionnalités de l'application en un nombre limité de grandes catégories de services que l'application doit offrir aux utilisateurs. Cela demande de la part du modélisateur non seulement une grande capacité de synthèse mais aussi une connaissance profonde du rôle joué par chaque utilisateur et des services attendus de l'application par ces derniers. La réalisation du prototype visuel avec les utilisateurs est l'occasion rêvée pour le modélisateur de comprendre leur rôle et leurs besoins, tout en dégageant les fonctionnalités de l'application. Une fois le prototype complété, il aura une vue d'ensemble des fonctionnalités attendues et il pourra en tirer un découpage tenant compte notamment des utilisateurs.

Le modélisateur a dégagé cinq grandes catégories de fonctionnalités dont il a fait cinq cas d'utilisation. La rédaction des scénarios va permettre d'exprimer ce que le prototype visuel ne peut faire voir, à savoir :

- Quel utilisateur est habilité à exploiter une fonctionnalité via un ou des panoramas dessinés plus tôt ;

- Quelles règles de gestion sont appliquées lorsqu'une fonctionnalité est exploitée ;
- La nature de l'interaction entre l'utilisateur et l'application, notamment le traitement des cas d'exception et des alternatives ;
- Les données générées par l'application qui ne sont pas explicitement présentes dans les divers panoramas mais dont il faudra assurer la persistance pour le bon fonctionnement de l'application.

Le diagramme de cas d'utilisation produit par le modélisateur est le suivant :



Chaque cas d'utilisation doit faire l'objet de scénarios qui décrivent en détail la chronologie d'événements susceptibles de se produire lorsque l'acteur engage une session de travail avec l'application dans le contexte particulier du cas d'utilisation. Ce qui peut donner lieu à un scénario principal et à un certain nombre de scénarios alternatifs, selon les services exigés de l'acteur.

Les scénarios qui suivent ont été rédigés par le modélisateur pour chaque cas d'utilisation. Certains scénarios comportent un scénario principal et des scénarios alternatifs. Tous débutent par une condition qui déclenche le cas d'utilisation ou qui justifie le scénario alternatif.

Scénarios	Cas d'utilisation : *Inscription des membres*
Principal	Un membre du personnel administratif désire procéder à l'inscription d'un membre. 1- Le panorama **Dossier membre** est activé. 1.a- S'il s'agit d'un nouveau membre, il saisit les coordonnées du membre. Le numéro du membre est généré par le système. 1.b- Sinon, à l'aide du bouton **Rechercher membre**, il recherche son dossier actuel sur la base de son nom et de son prénom ou uniquement sur la base de son numéro de membre. 2- Le système affiche les coordonnées du membre et ses dernières inscriptions. 2.a- Si les coordonnées du membre ont changé, le personnel les met à jour. 3- Il complète une inscription pour l'année en cours en saisissant le statut du membre, Athlète ou Bénévole, pour l'année en cours. 4- Les frais à acquitter, 0 $ ou 25 $ selon le statut, s'affichent ainsi que le solde à payer. 5- Pour finaliser l'inscription, le membre doit effectuer un paiement pour le solde à payer à son dossier. Le paiement génère une recette en date du jour, sa **Catégorie de recette** est « Inscription » et la valeur « Non » est générée pour la donnée **Recette transférée ?** indiquant que la recette n'a pas été transférée au système comptable. 5.a- Si le solde est à zéro après saisie du montant payé, le bouton **Imprimer carte membre** est disponible et l'impression de la carte du membre peut être faite. L'impression de la carte génère une valeur Oui pour la donnée **Carte imprimée ?**. Le dossier du membre passe à Actif. 5.b- Si le solde est supérieur à zéro, l'état du dossier reste Inactif, mais l'inscription est conservée.
Alternatif : Le membre est un athlète, il a le statut Actif mais souhaite faire le choix d'une nouvelle spécialité	3- Le panorama **Dossier athlète** est activé avec le bouton **Consulter dossier athlète** qui est alors disponible. La donnée **Sexe** est saisie le cas échéant. 4- La zone déroulante **Discipline** permet de consulter les disciplines associées à l'athlète. Si une discipline est choisie dans la liste, les spécialités de cette discipline, auxquelles l'athlète est déjà associé, peuplent la zone déroulante **Spécialité**. 5- Le choix d'une spécialité dans la liste affiche les résultats et les classements globaux et sous les onglets les résultats et les classements détaillés. 5.a- La zone déroulante **Spécialité** comporte aussi la mention « Nouvelle spécialité », qui lorsque choisie ouvre une fenêtre comportant toutes les disciplines et leurs spécialités. Le choix d'une spécialité ajoute au dossier de l'athlète cette spécialité pour laquelle tous les résultats et classements globaux sont initialement à zéro.

Alternatif : Le dossier du membre est inactif. Il possède une inscription pour l'année en cours et souhaite régler le solde par un paiement final	3- Le montant du solde payé est saisi et le solde est mis à zéro. Le paiement génère une recette en date du jour. Sa **Catégorie de recette** est « Inscription » et la valeur « Non » est générée pour la donnée **Recette transférée ?** indiquant que la recette n'a pas été transférée au système comptable. 4- Le bouton **Imprimer carte membre** est disponible et l'impression de la carte du membre peut être faite. L'impression de la carte génère une valeur « Oui » pour la donnée **Carte imprimée ?**. Le dossier du membre passe à Actif. Alternatif : Le personnel souhaite transférer au système comptable toutes les recettes et les dépenses compilées depuis le dernier transfert 6- Le bouton **Transférer au système informatisé de comptabilité** est activé. Toutes les recettes dont **Recette transférée ?** est « Non » et toutes les dépenses dont **Dépense transférée ?** est « Non » sont stockées dans un fichier pour transfert au système comptable. La valeur de la donnée est changée à « Oui » pour toutes les recettes et dépenses stockées dans le fichier. 7- Toutes les recettes et dépenses stockées dans le fichier sont soumises au système informatisé de comptabilité et une donnée est générée lors du transfert **Date de transfert recette** ou **Date de transfert dépense**.

Les scénarios décrivent clairement les règles de gestion de l'inscription des membres. Ils mettent en évidence de nouvelles données dont l'application devra assurer la persistance : **Carte imprimée ?**, **Date de transfert recette**, **Date de transfert dépense**, **Recette transférée ?**, et **Dépense transférée ?**. De plus, ils spécifient de plus la séquence des interactions entre l'utilisateur et le système, séquence qui constitue un énoncé des spécifications de l'interface utilisateur de l'application à réaliser.

Chaque élément d'une séquence d'interactions porte un numéro. Si le même numéro de séquence est présent dans un scénario alternatif, cet élément est substitué à celui présent dans le scénario principal. Si un numéro est supérieur au dernier numéro présent dans le scénario principal, l'élément s'ajoute au scénario principal (voir à cet effet l'élément 6 du dernier scénario alternatif de **Inscription des membres**).

Scénarios	Cas d'utilisation : *Organisation des activités*
Principal	Le directeur d'un comité d'organisation souhaite inscrire une nouvelle activité ou modifier les données associées à une activité. 1- Le panorama **Activité et comité d'organisation** est activé. 1.a- S'il s'agit d'une nouvelle activité, il saisit les données sur l'activité dont obligatoirement le **Titre de l'activité** et les données sur le directeur. Les membres du comité et leur responsabilité dans le cadre de l'activité peuvent être saisis. Le numéro de l'activité est généré par le système. 1.b- Sinon, il saisit un numéro d'activité qui donne accès aux données existantes sur l'activité. 2- Le membre du comité est choisi à partir d'une liste des membres actifs qui s'affiche dès que le pointeur est placé dans le champ **No membre**. 3- La responsabilité du membre est saisie à partir d'une liste prédéfinie fournie par le système. 4- Le bouton **Détails de l'activité** ouvre une fenêtre dont le contenu est déterminé par le **Type d'activité**.
Alternatif : L'activité est du type Financement	5- Le panorama **Détails activités de financement** est activé. 6- La date, l'heure et l'adresse où se tient l'activité doivent être saisies ou peuvent être modifiées.
Alternatif : l'activité est du type Compétition interne	5- Le panorama **Détails compétitions interne** est activé. 6- Les coordonnées de l'activité sont saisies ou modifiées à loisir. 7- L'onglet **Épreuves au programme** doit être activé. Les épreuves aux programmes sont saisies en fournissant obligatoirement la date, l'heure, le lieu, la discipline et la spécialité de l'épreuve. Il est possible de naviguer à travers les éléments du programme pour consultation ou modification grâce aux boutons de navigation. Le bouton **Imp. Programme** commande l'impression du programme de l'activité incluant les participants le cas échéant.
Alternatif : l'activité est du type Compétition externe	5- Le panorama **Détails compétitions externe** est activé. 6- Les coordonnées de l'activité sont saisies ou modifiées à loisir. 7- Le bouton **Programme et logistique** active le panorama **Détails compétitions externe (programme et logistique)** 8- Les coordonnées de l'activité sont affichées et les onglets **Épreuves au programme** et **Logistique** donnent accès à des données qui peuvent faire l'objet de saisie ou de modification. 8.a- Si l'onglet **Épreuves au programme** est activé, les épreuves aux programmes doivent être saisies en fournissant obligatoirement la date, l'heure, le lieu, la discipline et la spécialité de l'épreuve. Il est possible de naviguer à travers les éléments du programme pour consultation ou modification grâce aux boutons de navigation. Le bouton **Imp. Programme** commande l'impression du programme incluant les participants le cas échéant. 8.b- Si l'onglet **Logistique** est activé, les accompagnateurs, le logement et le transport relatifs à l'activité sont saisis ou modifiés. Les accompagnateurs sont choisis à partir d'une liste des membres actifs qui s'affiche à l'ajout d'un accompagnateur. La responsabilité de l'accompagnateur est saisie à partir d'une liste prédéfinie fournie par le système.

Scénarios	Cas d'utilisation : *Inscription aux compétitions*
Principal	Le personnel reçoit une demande d'inscription à une compétition et souhaite inscrire l'athlète à une épreuve. 1- Le panorama **Activité et comité d'organisation** est activé. Il saisit un numéro d'activité qui donne accès aux données existantes sur l'activité. L'activité doit être une compétition. 2- Le bouton **Détails de l'activité** permet d'activer le panorama **Détails compétitions interne** ou **Détails compétitions externe (programme et logistique)** selon qu'il s'agit d'une compétition interne ou externe. L'onglet **Épreuves au programme** doit être activé. 3- Le personnel repère l'épreuve grâce aux boutons de navigation.
Alternatif : l'activité est du type Compétition interne	4- Le participant est choisi à partir d'une liste d'athlètes actifs dans la spécialité de l'épreuve. Celle-ci s'affiche dès que le pointeur est placé dans le champ **No athlète**. 5- Les champs **Résultat** et **Rang** sont laissés en blanc.
Alternatif : l'activité est du type Compétition externe	4- Le participant est choisi à partir d'une liste des athlètes déjà inscrits à une autre épreuve de la compétition et qui s'affiche dès que le pointeur est placé dans le champ **No athlète**. 4.a- Si l'athlète n'est pas dans la liste, le **Nom** et le **Prénom** du participant sont saisis explicitement. Un **Numéro** est attribué automatiquement par le système à chaque nouveau participant.

Scénarios	Cas d'utilisation : *Compilation des recettes et des dépenses*
Principal	Le personnel souhaite saisir une recette ou une dépense dans le cadre d'une activité. 1- Le panorama **Activité et comité d'organisation** est activé. Il saisit un numéro d'activité qui donne accès aux données existantes sur l'activité. 2- Le bouton **Détails de l'activité** permet d'activer le panorama **Détails activité de financement**, **Détails compétitions interne** ou **Détails compétitions externe (programme et logistique)** selon le type d'activité. L'onglet **Recettes** ou **Dépenses** doit être activé, selon le type de transaction à saisir.
Alternatif : la transaction est une recette	3- La date de la recette est saisie. 4- La catégorie de recette est choisie à partir d'une liste prédéfinie. 5- Le montant de la recette est saisi. 6- La valeur "Non" est générée pour la donnée **Recette transférée ?** associée à la recette.
Alternatif : la transaction est une dépense	3- La date de la dépense est saisie. 4- La catégorie de dépense est choisie à partir d'une liste prédéfinie. 5- Le montant de la dépense est saisi. 6- La valeur "Non" est générée pour la donnée **Dépense transférée ?** associée à la dépense.

Scénarios	Cas d'utilisation : *Compilation des résultats*
Principal	Un entraîneur souhaite inscrire un résultat à une épreuve ou un résultat à l'entraînement. 1.a- S'il s'agit d'un résultat à une épreuve, le panorama **Activité et comité d'organisation** est activé. Il saisit un numéro d'activité qui donne accès aux données existantes sur l'activité. L'activité doit être une compétition. Le bouton **Détails activité** est activé. 1.b- S'il s'agit d'un résultat à l'entraînement, le panorama **Dossier membre** est activé. Le bouton **Consulter dossier athlète** permet d'activer le panorama **Dossier Athlète**. Lorsque l'entraîneur choisit la spécialité dans laquelle le résultat doit être compilé, les champs **Résultat record** et **Moyennes résultats** sont calculés sur la base de tous les résultats de cet athlète dans cette spécialité.
Alternatif : Le résultat est obtenu en compétition	2- L'onglet **Épreuves au programme** est activé. 3- L'entraîneur repère l'épreuve grâce aux boutons de navigation. 4- À la suite du nom de l'athlète, le résultat, l'unité de mesure et le rang sont saisis.
Alternatif : Le résultat est obtenu à l'entraînement	2- L'onglet **Entraînement** est activé. 3- La date du résultat, le résultat, l'unité de mesure sont saisis. 4- Les champs **Résultat record** et **Résultat moyen** sont calculés à nouveau pour tenir compte du nouveau résultat de cet athlète dans cette spécialité.

Le scénario **Compilation des résultats** indique que les données **Résultat record** et **Résultat moyen** sont calculées à chaque fois qu'un résultat est saisi. On en déduit bien évidemment qu'il s'agit d'une donnée calculée, mais il n'y a pas lieu d'assurer sa persistance.

Bien que l'exercice d'écriture des scénarios peut sembler long et même fastidieux, il ne s'agit pas d'un exercice futile. Ce cas montre nettement qu'il permet de recenser des données non présentes dans les panoramas mais dont la persistance doit être assurée (**Carte imprimée ?**, **Date de transfert** et **Transférée ?**). De plus certaines données calculées présentes dans les panoramas pourraient ne pas être considérées comme persistantes en vertu de leur mode de génération (Exemple : **Résultat record** et **Résultat moyen**).

Le modélisateur peut maintenant procéder au recensement formel des données persistantes à partir des descriptifs des documents tirés des panoramas et des scénarios de cas d'utilisation qu'il a par ailleurs rédigés. Le format adopté est en tout point conforme à celui employé pour présenter les données persistantes tirées d'un modèle de fonctionnement d'un système d'information. Le feuille de calcul suivante représente le recensement définitif des données persistantes du cas 4-5.

RECENSEMENT DES DONNÉES

Projet : Gestion des activités du club Date : 16/3/2007

Nom donnée	P	Type	Mode	Contrainte d'intégrité	Règle de calcul
Dossier membre					
Numéro membre	X	entier	mémorisé		
Prénom membre	X	texte	mémorisé		
Nom membre	X	texte	mémorisé		
NAS membre	X	texte	mémorisé		
Date de naissance membre	X	date	mémorisé		
Adresse membre	X	texte	mémorisé		
Téléphone domicile	X	texte	mémorisé		
Téléphone bureau	X	texte	mémorisé		
Année inscription	X	texte	mémorisé		
Statut membre	X	texte	mémorisé	Athlète ou Bénévole	
Frais inscription	X	monnaie	mémorisé		
Montant payé	X	monnaie	mémorisé		
Solde à payer	X	monnaie	mémorisé		
Dossier athlète					
Âge membre		entier	calculé		
État du dossier	X	texte	mémorisé	Actif ou Inactif	
Sexe membre	X	texte	mémorisé	Féminin ou Masculin	
Nom discipline	X	texte	mémorisé		
Nom spécialité	X	texte	mémorisé		
Résultat record		réel	calculé		
Unité mesure record		texte	mémorisé	Sec., mètre, min., h.	
Résultat moyen		réel	calculé		
Unité mesure moyen		texte	mémorisé	Sec., mètre, min., h.	
Classement national	X	entier	mémorisé		
Classement provincial	X	entier	mémorisé		
Date résultat	X	date	mémorisé		
Rang	X	entier	mémorisé		
Résultat	X	réel	mémorisé		
Unité mesure résultat	X	texte	mémorisé	Sec., mètre, min., h.	
Type résultat	X	texte	mémorisé	Entraînement ou Compétition	

Nom donnée	P	Type	Mode	Contrainte d'intégrité	Règle de calcul
Activité et comité					
Numéro activité	X	entier	mémorisé		
Titre activité	X	texte	mémorisé		
Type d'activité	X	texte	mémorisé		
Responsabilité	X	texte	mémorisé	Juge, Secrétaire/ trésorier, Entraîneur, Conducteur, Chef de mission, Directeur activité	
Détails activité de financement					
Date activité	X	date	mémorisé		
Heure activité	X	date	mémorisé		
Adresse activité	X	texte	mémorisé		
Date recette	X	date	mémorisé		
Catégorie recette	X	texte	mémorisé	Inscription, Don, Commandite	
Montant recette	X	monnaie	mémorisé		
Date dépense	X	date	mémorisé		
Catégorie dépense	X	texte	mémorisé	Location local, Personnel, Location véhicule, Logement, Sécurité	
Montant dépense	X	monnaie	mémorisé		
Recette brute		monnaie	calculé		
Dépenses totales		monnaie	calculé		
Recette nette		monnaie	calculé		
Détails compétition externe					
Date début compétition	X	date	mémorisé		
Date fin compétition	X	date	mémorisé		
Lieu compétition	X	texte	mémorisé		
Nom organisateur	X	texte	mémorisé		
Adresse organisateur	X	texte	mémorisé		
Nom responsable	X	texte	mémorisé		
Prénom responsable	X	texte	mémorisé		
Téléphone responsable	X	texte	mémorisé		
Télécopieur responsable	X	texte	mémorisé		

Nom donnée	P	Type	Mode	Contrainte d'intégrité	Règle de calcul
Détails compétition interne					
Date épreuve	X	date	mémorisé		
Heure épreuve	X	date	mémorisé		
Local épreuve	X	réel	mémorisé		
Détails compétition externe (programme et logistique)					
Nom établissement	X	texte	mémorisé		
Adresse établissement	X	texte	mémorisé		
Nombre de chambres	X	entier	mémorisé		
Immatriculation véhicule	X	texte	mémorisé		
Locateur véhicule	X	texte	mémorisé		
Adresse locateur	X	texte	mémorisé		
Nombre de passagers	X	entier	mémorisé		
Autres données persistantes					
Carte imprimée ?	X	booléen	mémorisé		
Recette transférée ?	X	booléen	mémorisé		
Date de transfert recette	X	date	mémorisé		
Dépense transférée ?	X	booléen	mémorisé		
Date de transfert dépense	X	date	mémorisé		

Le cas 4-5 fait ressortir à la fois les différences et les similitudes entre un modèle de fonctionnement d'un système d'information et un modèle de fonctionnement d'une application de bases de données. Considérons à tour de rôles les similitudes et les différences.

La nature des similitudes

1. Chaque modèle vise le même objectif : le recensement des données persistantes pour assurer le fonctionnement d'une application de base de données. Dans un cas il s'agit d'une application visant à automatiser un système d'information, dans l'autre il s'agit d'une application originale qui ne peut s'appuyer sur un système d'information existant.
2. Chaque modèle comporte une représentation des documents que le modélisateur identifie comme la source première de recensement des données persistantes. Cette représentation prend la forme d'un ensemble de descriptifs de documents, dans un cas il s'agit de documents papier dans l'autre de documents électroniques.
3. Chaque modèle fait appel à un diagramme de cas d'utilisation illustrant le découpage des fonctionnalités de l'application à réaliser. Un tel découpage va permettre de planifier les étapes de réalisation de l'application en priorisant la réalisation de certains cas d'utilisation avant d'autres.
4. Les scénarios de cas d'utilisation mettent en évidence des règles de gestion. Ils réfèrent de plus à des données persistantes qui n'apparaissent pas dans les descriptifs des documents mais sont jugées vitales au fonctionnement de l'application.

La nature des différences

1. Dans le modèle de fonctionnement d'un système d'information, les cas d'utilisation permettent d'identifier les documents papier où seront recensées les données persistantes. Dans le modèle de fonctionnement de l'application, le prototype visuel fait état des documents électroniques de l'application et les cas d'utilisation ne sont réalisés qu'après avoir procédé au dessin des panoramas du prototype visuel.
2. Les scénarios de cas d'utilisation d'un système d'information mettent en cause des acteurs principaux représentant surtout des personnes extérieures à l'organisation. Les scénarios de cas d'utilisation d'une application mettent en cause les utilisateurs de l'application qui en général sont des personnes appartenant à l'organisation.

3. Les scénarios de cas d'utilisation d'un système d'information décrivent les échanges de données entre les acteurs et le système, à un niveau assez général, sans spécifier les détails de nature technique. Les scénarios des cas d'utilisation d'une application décrivent la nature des interactions entre les acteurs et l'application. Ils sont en conséquence beaucoup plus concrets et font souvent référence aux objets d'interaction présents dans un panorama (bouton, zone de liste déroulante, menu, zone de saisie, etc.). Ils font état des données que ces objets permettent de consulter ou de saisir. Ils sont formulés avec un souci du détail beaucoup plus prononcé. Ces scénarios seront à nouveau exploités durant la phase de conception et de réalisation car ils décrivent les fonctionnalités de l'application sur le plan technique.

Réalisation du modèle conceptuel de données sur la base du recensement des données persistantes

La tâche de réaliser un modèle conceptuel de données peut s'avérer ardue lorsque le recensement des données persistantes génère une grande quantité de données et que par ailleurs les descriptifs de documents, qui illustrent la nature des associations entre ces données, sont volumineux.

Aussi faut-il s'attaquer à cette tâche de manière systématique. Nous renvoyons le lecteur au chapitre 1 qui prescrit des règles et des astuces pour la modélisation conceptuelle des données. On y présente au tableau 1-3 des catégories générales auxquelles peuvent appartenir les entités d'un modèle conceptuel de données lorsque ce modèle est destiné à représenter les données traitées dans une organisation. Ces catégories permettent de proposer une démarche de modélisation conceptuelle de données, tirées d'un recensement de données persistantes.

La proposition consiste à faire de chaque donnée persistante un attribut exclusif d'une entité. Suivant en cela les prescriptions de la méthode DATARUN, nous suggérons de procéder dans l'ordre suivant :

1. Identifier des entités représentant des biens ou des services offerts par l'organisation ;
2. Identifier des entités présentes dans l'environnement de l'organisation incluant des systèmes externes ;
3. Identifier des entités représentant des ressources internes à l'organisation nécessaires aux opérations de l'organisation ; il peut s'agir de ressources humaines, matérielles ou financières, assimilées à des composants du système opérant ;

4. Identifier des entités représentant des transactions entre le système opérant avec des entités de l'environnement de l'organisation ;
5. Identifier des entités représentant les décisions prises dans l'organisation ;
6. Identifier des entités représentant des transactions à l'intérieur du système opérant.

À chaque fois que le modélisateur fait ressortir une entité, il doit choisir, parmi les données persistantes, un attribut qui peut agir comme identifiant de l'entité et, si un tel identifiant naturel n'existe pas, il devra créer un identifiant artificiel. Rappelons cependant qu'une entité n'a pas obligatoirement un identifiant explicite. En effet, une entité liée à une autre par une association spécialisée, héritage ou composition, possède un identifiant implicite. Il en va de même pour les entités d'association.

Les attributs autres que l'identifiant seront choisis parmi les données persistantes et devront dépendre fonctionnellement et exclusivement de leur identifiant lorsque ce dernier est connu. Une donnée persistante ne peut être rattachée qu'à une seule entité (règle de non redondance).

Dès lors qu'un certain nombre d'entités ont été identifiées, il y a lieu d'étudier les associations entre les entités sur la base des descriptifs des documents. Les descriptifs montrent les liens *un à un* ou *un à plusieurs* entre les données. Ils permettent en conséquence de visualiser les associations possibles entre les entités qui regroupent ces données et de fixer leur multiplicité maximale (un ou plusieurs). Il est normal que certaines données persistantes ne puissent être rattachées à une entité dérivée de la démarche proposée. C'est le cas des données qui dépendent fonctionnellement de deux ou de plusieurs entités à la fois. Dans ce cas, elles devront être rattachées à une entité d'association.

Nous proposons d'illustrer la démarche, dont les grandes lignes ont été données ci-dessus, à l'aide d'études de cas qui s'inscrivent dans la continuité de ceux discutés au cours de ce chapitre.

CAS 4-6 MODÈLE CONCEPTUEL DU CAS TANNERIE TANBEC

Considérant les **Descriptifs de document** et le **Recensement des données persistantes** produits au cours de l'étude de cas 4-3, élaborer un modèle conceptuel de données en suivant une démarche systématique basée sur les catégories d'entités.

1. Catégorie Produits et services.

TANBEC offre des services de tannage. Le descriptif du document **Tarification** fait ressortir la nature de ses services et leur prix. Il y a lieu de regrouper sous une entité TYPE DE TANNAGE les données de ce descriptif.

2. Catégorie Entités dans l'environnement de l'organisation.

Les entités de cette catégorie sont généralement les acteurs du diagramme de cas d'utilisation dans le modèle de fonctionnement du système d'information. Attention ! ce n'est pas le cas des acteurs d'un modèle de fonctionnement de l'application qui sont des utilisateurs de l'application, donc généralement des ressources humaines interne à l'organisation.

Le diagramme de cas d'utilisation du cas TANBEC propose deux acteurs qui seront considérés comme des objets de l'environnement : **Client** et **Fournisseur**.

3. Catégorie Composant du système opérant.

Les données persistantes ne reflètent aucun attribut d'une ressource interne à l'organisation. Par exemple, aucun document et aucun cas d'utilisation ne fait référence aux personnes impliquées dans la prise d'une commande, son exécution et sa facturation.

4. Catégorie Transactions avec des entités de l'environnement.

Ce cas comporte de nombreuses entités de ce type dont les commandes, les factures et les paiements. Nous relevons donc les entités **Commande client** et **Commande fournisseur**, **Facture client** et **Facture fournisseur**, **Paiement client**.

Il s'agit de transactions effectuées avec l'une ou l'autre des entités de l'environnement : **Client** et **Fournisseur**.

Cette fois, il y a lieu d'associer les transactions avec l'entité externe qui est en l'objet, soit directement, soit indirectement tout en évitant les associations redondantes. Les associations directes avec les entités externes sont les suivantes :

Commande client- Client
Commande fournisseur- Fournisseur
Facture fournisseur- Fournisseur

L'association suivante découlant du descriptif FACTURE CLIENT :
Facture client- Commande client
donne lieu à une association indirecte de **Facture client** à **Client** ;

L'association suivante découlant du descriptif REÇU :
Paiement client- Facture client
donne lieu à une association indirecte de **Facture client** à **Client** ;

Il n'y a pas lieu d'établir une association directe entre l'entité **Client** et les entités **Facture client** et **Paiement client** comme le montre le descriptif DOSSIER CLIENT, celles-ci étant assurées indirectement.

En terminant, il nous faut formuler les associations qui lient une transaction avec d'autres entités que les entités externes. Le descriptif COMMANDE CLIENT montre une association entre une **Commande client** et le **Type de tannage**. De plus le descriptif COMMANDE AU FOURNISSEUR montre une association **Commande fournisseur-Commande client.**

5. Catégorie Décisions prises dans l'organisation.

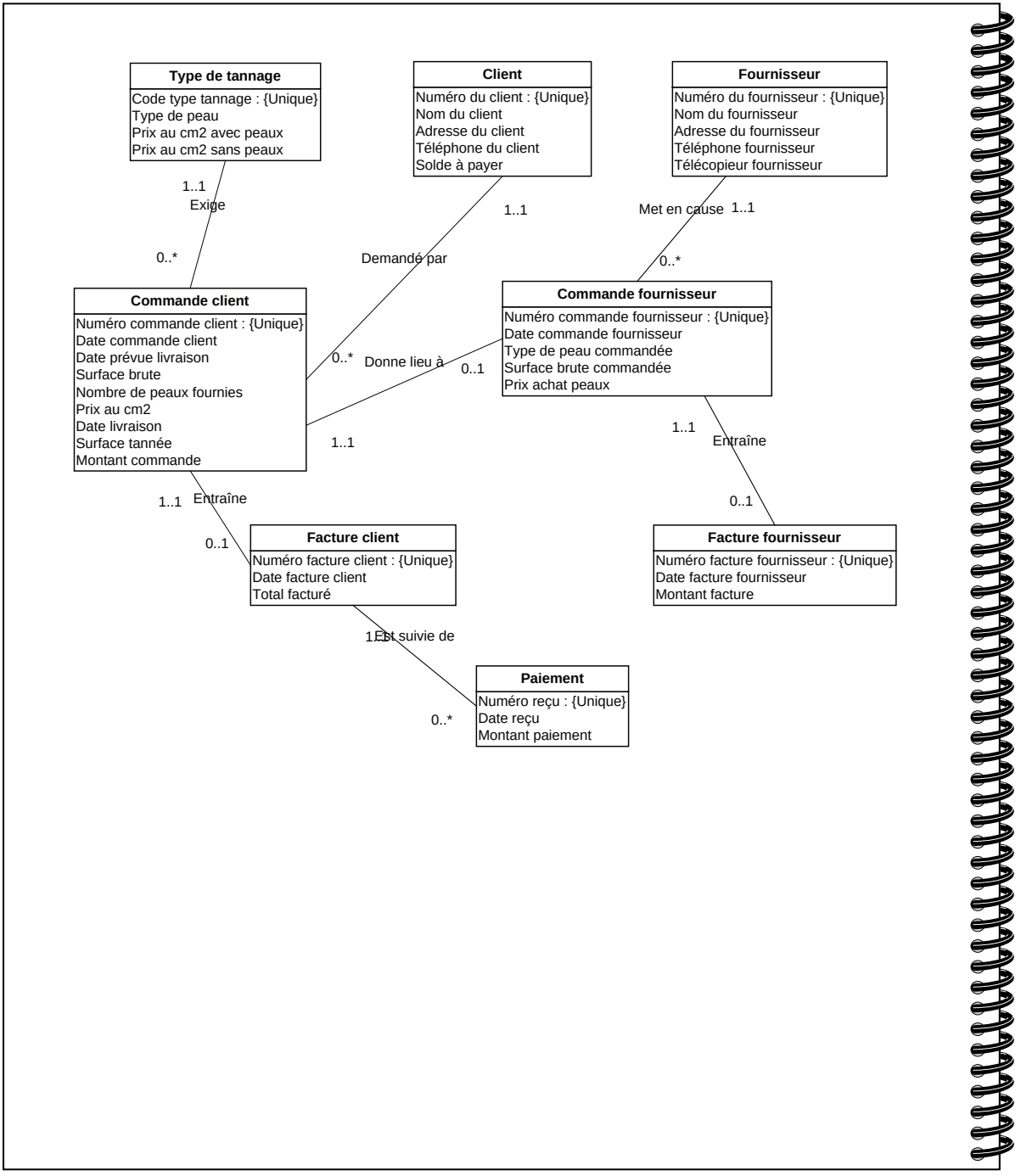
Bien que le commis traitant la commande d'un client doive décider d'y donner suite ou non selon l'état du dossier du client, cette décision ne donne pas lieu à une donnée persistante.

6. Catégorie Transactions internes au système opérant.

La **Surface tannée** est une donnée dont la valeur est générée dans le cadre de l'exécution de la commande. Il s'agit donc d'une donnée qui devrait être rattachée à une entité de type transaction interne regroupant des attributs décrivant le résultat d'une opération interne. Puisque **Surface tannée** serait le seul attribut d'une telle transaction selon le recensement des données persistantes, le modélisateur ne juge pas pertinent de créer une nouvelle entité appelée **Exécution commande** ; il propose donc de rattacher son seul attribut potentiel à **Commande client**.

La donnée **Date livraison** dont la valeur est générée dans le cadre de la livraison de la commande est aussi le résultat d'une transaction interne. Le modélisateur ne juge pas pertinent de créer une entité **Livraison** ne comportant que l'attribut **Date livraison**. Il choisira plutôt de rattacher cet attribut à **Commande client**.

Considérant les entités et les associations qui ont été identifiées en suivant la démarche proposée, il reste au modélisateur à faire le choix de leur identifiant, ici tous des identifiants naturels, et de rattacher les autres données persistantes à une entité. Les données persistantes peuvent être rattachées à l'une ou l'autre des entités identifiées plus tôt et elles possèdent toutes une dépendance fonctionnelle à leur identifiant. Aucune entité d'association n'est alors requise.


Type de tannage
Code type tannage : {Unique}
Type de peau
Prix au cm2 avec peaux
Prix au cm2 sans peaux
Client
Numéro du client : {Unique}
Nom du client
Adresse du client
Téléphone du client
Solde à payer
Fournisseur
Numéro du fournisseur : {Unique}
Nom du fournisseur
Adresse du fournisseur
Téléphone fournisseur
Télécopieur fournisseur
1..1
Exige
0..*
1..1
Demandé par
0..*
Met en cause
1..1
0..*
Commande client
Numéro commande client : {Unique}
Date commande client
Date prévue livraison
Surface brute
Nombre de peaux fournies
Prix au cm2
Date livraison
Surface tannée
Montant commande
0..*
Donne lieu à
0..1
Commande fournisseur
Numéro commande fournisseur : {Unique}
Date commande fournisseur
Type de peau commandée
Surface brute commandée
Prix achat peaux
1..1
1..1
Entraîne
0..1
1..1
Entraîne
0..1
Facture client
Numéro facture client : {Unique}
Date facture client
Total facturé
Facture fournisseur
Numéro facture fournisseur : {Unique}
Date facture fournisseur
Montant facture
1
Est suivie de
0..*
Paiement
Numéro reçu : {Unique}
Date reçu
Montant paiement

CAS 4-7 MODÈLE CONCEPTUEL DU CAS CLUB DE VOILE

Considérant les **Descriptifs de document** et le **Recensement des données persistantes** produits au cours de l'étude de cas 4-4, élaborer un modèle conceptuel de données en suivant une démarche systématique basée sur les catégories d'entités.

1. Catégorie Produits et services.

Le club de voile se spécialise dans l'organisation de régates, saison après saison. Les entités dans cette catégorie **Régate** et **Étape** seront associées par une composition, une régate étant constituée d'étapes qui lui appartiennent de manière exclusive.

2. Catégorie Entités dans l'environnement de l'organisation.

Les acteurs du diagramme de cas d'utilisation sont au nombre de trois : Club associé, FVQ et Skipper. Le recensement ne fait ressortir aucune donnée qui pourrait être un attribut de ces acteurs, sauf pour l'acteur Skipper. Or Skipper est un rôle joué par un membre d'équipage et il n'y a pas lieu d'en faire une entité en soi. L'entité de l'environnement la plus pertinente est **Membre équipage** dont skipper est un type particulier.

Par ailleurs, le recensement souligne des données sur les voiliers qui sont aussi nettement des objets de l'environnement du Club. On en fera une entité appelée **Voilier**. Cette entité devra être associée à **Régate** pour mémoriser la participation du voilier à une régate.

De plus **Membre équipage** devra être associé à **Étape** pour mémoriser la participation d'un membre à une étape en particulier. Cette association devra être réalisée grâce à une entité d'association, **Participation**, qui aura comme attribut la donnée persistante **Statut membre** car elle dépend fonctionnellement de **Membre équipage** et **Étape** à la fois. **Participation** sera de plus associée à **Voilier** pour mémoriser le voilier sur lequel la participation s'exerce.

3. Catégorie Composant du système opérant.

Il doit s'agit ici d'une entité représentant une ressource interne à l'organisation directement impliquée dans ses opérations. Le **Juge** est dans cette catégorie. Il compile les résultats, reçoit les réclamations et sanctionne les participants. La **Saison** et la **Classe** constituent des entités internes requises pour mener les opérations.

Dès lors, il est possible d'associer ces nouvelles entités. **Juge** doit être associé à **Étape** à deux titres : mémoriser la disponibilité du juge pour une étape et mémoriser l'affectation officielle du juge à une étape. Deux associations devront être mises en œuvre.

Régate doit être associée à **Saison** pour mémoriser dans quelle saison la régate est tenue. Enfin **Voilier** doit être associé à **Saison** pour mémoriser les résultats du voilier au cours des saisons de régate.

4. Catégorie Transactions avec des entités de l'environnement.

Il doit s'agir de transactions effectuées avec l'une ou l'autre des entités de l'environnement : soit **Membre équipage** ou **Voilier**. Bien que certaines transactions mettent en cause le **Voilier**, le système traite uniquement avec l'entité **Membre équipage**. L'entité **Inscription** est de ce type. Elle doit être associée à **Membre équipage** pour mémoriser le skipper qui procède à une inscription.

Il faut aussi associer **Inscription** avec **Voilier** et **Inscription** avec **Classe** car une inscription met en cause un voilier dans une classe de participation donnée.

Tout changement à un équipage donne lieu à une transaction avec le skipper qui fait connaître son nouvel équipage à une étape donnée. L'entité d'association Participation mémorise la participation d'un membre d'équipage à une étape et donc tout changement apporté à un équipage à une étape donnée. Elle peut être considérée comme une transaction externe et les attributs **Date changement équipage** et **Heure changement équipage** peuvent lui être rattachés.

Le paiement donne lieu à une transaction distincte de l'inscription ce qui justifie l'entité **Paiement** avec des attributs qui lui sont propres. Elle sera évidemment associée à **Inscription**.

Enfin, une réclamation doit être considérée comme une transaction externe. L'entité **Réclamation** sera associée à **Inscription** à deux titres : pour mémoriser le réclamant d'une part et les intimés d'autre part. **Réclamation** sera aussi associée à **Juge** et à **Étape** pour mémoriser le juge qui reçoit la réclamation et l'étape en litige.

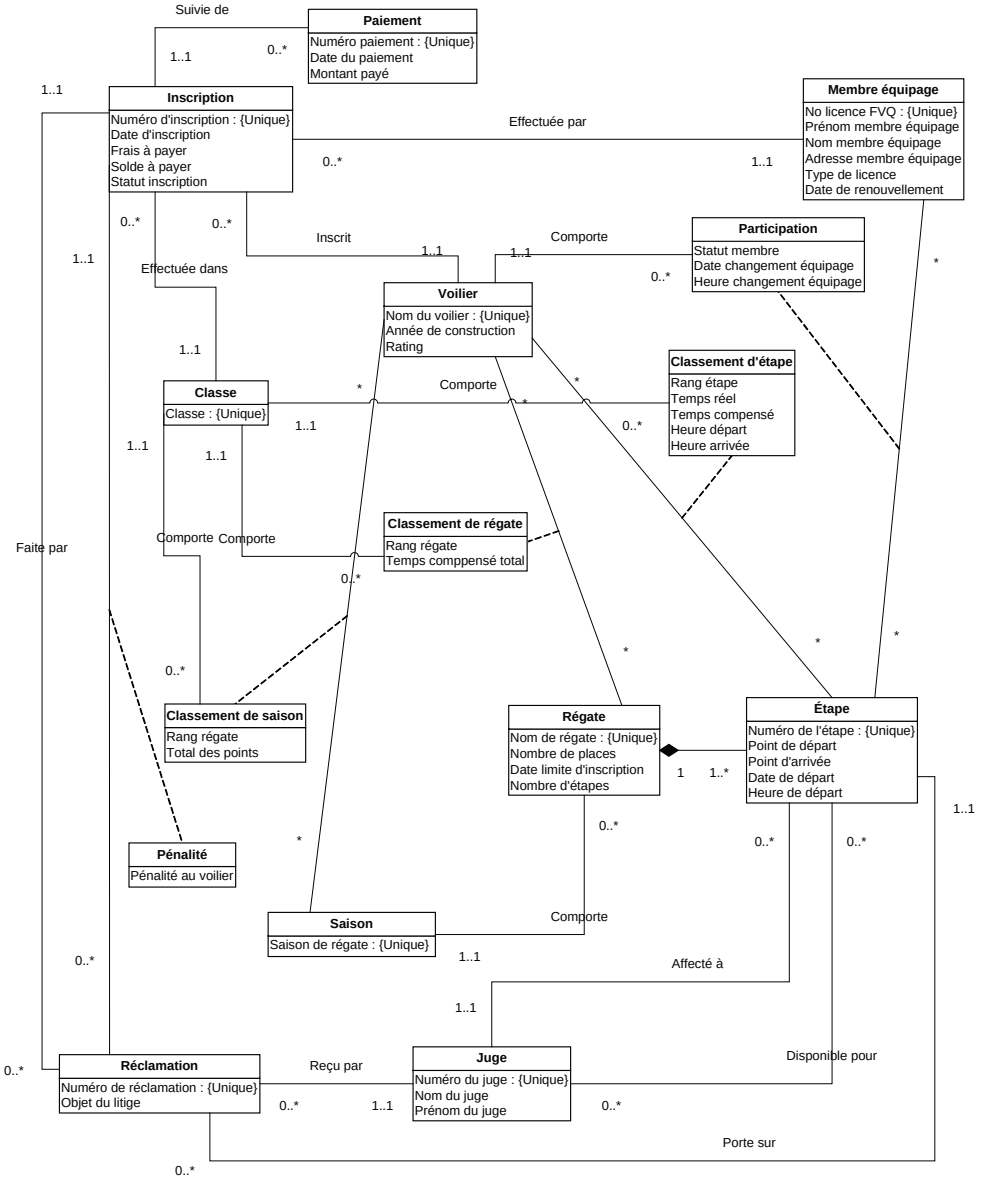
5. Catégorie Décisions prises dans l'organisation.

La décision du juge qui traite une réclamation fait l'objet d'une éventuelle pénalité pour un participant. L'entité **Pénalité** représente une décision à mémoriser. Elle sera considérée comme une entité d'association rattachée à **Réclamation-Inscription** qui mémorise les voiliers incriminés dans une réclamation, car une pénalité dépend fonctionnellement à la fois d'une inscription et d'une réclamation.

6. Catégorie Transactions internes au système opérant.

Chaque opération menant à l'établissement du rang d'un voilier dans une étape, dans une régate ou dans une saison donne lieu à une transaction interne. Nous les appellerons respectivement **Classement d'étape**, **Classement de régate** et **Classement de saison**. Ces entités seront des entités d'association rattachées respectivement à l'association **Voilier-Étape**, **Voilier-Régate** et **Voilier-Saison**.

La démarche mène à la production du modèle conceptuel suivant.


Paiement
Numéro paiement : {Unique}
Date du paiement
Montant payé
Suivie de
1..1
0..*
Inscription
Numéro d'inscription : {Unique}
Date d'inscription
Frais à payer
Solde à payer
Statut inscription
Effectuée par
Membre équipage
No licence FVQ : {Unique}
Prénom membre équipage
Nom membre équipage
Adresse membre équipage
Type de licence
Date de renouvellement
Inscrit
Comporte
Participation
Statut membre
Date changement équipage
Heure changement équipage
Effectuée dans
Voilier
Nom du voilier : {Unique}
Année de construction
Rating
Classe
Classe : {Unique}
Classement d'étape
Rang étape
Temps réel
Temps compensé
Heure départ
Heure arrivée
Classement de régate
Rang régate
Temps comppensé total
Faite par
Classement de saison
Rang régate
Total des points
Régate
Nom de régate : {Unique}
Nombre de places
Date limite d'inscription
Nombre d'étapes
Étape
Numéro de l'étape : {Unique}
Point de départ
Point d'arrivée
Date de départ
Heure de départ
Pénalité
Pénalité au voilier
Saison
Saison de régate : {Unique}
Affecté à
Réclamation
Numéro de réclamation : {Unique}
Objet du litige
Reçu par
Juge
Numéro du juge : {Unique}
Nom du juge
Prénom du juge
Disponible pour
Porte sur

CAS 4-8 MODÈLE CONCEPTUEL DU CAS CLUB D'ATHLÉTISME

Considérant les Descriptifs de document et le Recensement des données persistantes produits au cours de l'étude de cas 4-5, élaborer un modèle conceptuel de données en suivant une démarche systématique basée sur les catégories d'entités.

1. Catégorie Produits et services.

Le club d'athlétisme propose des activités de natures variées. Les attributs communs à toutes les activités sont regroupés dans l'entité **Activité**. Cette entité se spécialise en **Activité de financement** ou en **Compétition**. Ces dernières seront associées par un lien d'héritage à **Activité**. Les activités de compétition regroupent des épreuves. L'entité **Épreuve** s'impose pour décrire une activité de compétition, d'où une association de composition entre **Compétition** et **Épreuve**.

2. Catégorie Entités dans l'environnement de l'organisation.

Deux entités externes aux opérations du Club d'athlétisme ressortent : **Membre** et **Organisateur externe**. Au sens strict, le membre ne fait pas partie de l'organisation car il n'est pas en général impliqué dans les opérations du Club. Il est surtout le bénéficiaire de ses services. L'entité **Membre** sera associée à **Activité** pour mémoriser la responsabilité que le membre exerce dans une activité.

3. Catégorie Composant du système opérant.

Les véhicules et les unités de logement offerts aux membres pour des compétitions externes constituent des ressources du système opérant. **Logement** et **Véhicule** sont donc des entités pertinentes. Elles seront associées à **Compétition**. Les entités **Spécialité**, **Discipline** et **Responsabilité**, représentent des composants du système opérant car elles apportent une description des services offerts aux membres. **Spécialité** et **Responsabilité** devront être associées à **Membre** pour mémoriser une spécialité du membre athlète ou la responsabilité d'un membre dans une activité. **Spécialité** appartient à une **Discipline**, les deux entités devront donc être associées.

4. Catégorie Transactions avec des entités de l'environnement.

Inscription et **Paiement** constituent des entités représentant des transactions avec le membre. Elles seront associées directement ou indirectement à **Membre**.

5. Catégorie Décisions prises dans l'organisation.

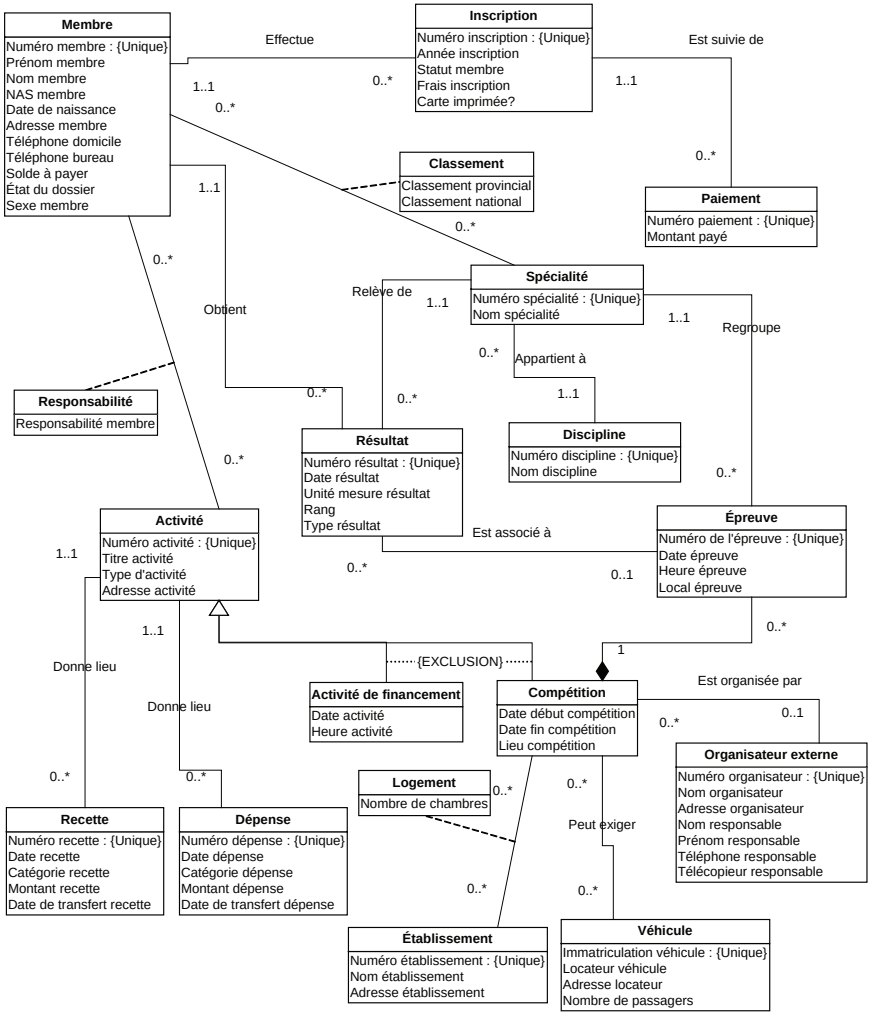
Aucune entité ne découle d'une décision prise au niveau des opérations.

6. Catégorie Transactions internes au système opérant.

Les entités **Dépense** et **Recette** représentent des transaction internes. Elles sont associées à une activité. Il en va de même des résultats obtenus par les athlètes. L'entité **Résultat**

s'impose comme transaction interne. Elle sera modélisée comme entité d'association entre **Membre** et **Spécialité** et elle sera de plus associée à l'entité **Épreuve**. Il en va de même pour l'entité **Classement** qui mémorise le classement du membre dans une spécialité à la fois au niveau provincial et au niveau national.

Le modèle conceptuel présenté ci-dessous dérive de la démarche suivie pour l'identification des entités pertinentes. Toutes les données persistantes ont par ailleurs été rattachées à titre d'attribut à l'une ou l'autre des entités.



Ces études de cas concluent le traitement de la phase d'analyse des besoins dans le contexte de la réalisation d'une application de bases de données. Rappelons que cette première phase consiste essentiellement à recenser les données persistantes et à les représenter par le biais d'un modèle conceptuel de données. Deux approches ont été présentées pour réaliser le recensement des données persistantes. Elles sont basées respectivement sur l'élaboration d'un modèle de fonctionnement du système d'information ou sur l'élaboration d'un modèle de fonctionnement de l'application. Pour assurer la représentation des données persistantes en réalisant un modèle conceptuel de données, nous avons exposé une démarche systématique qui s'appuie sur des catégories d'entités conceptuelles, démarche largement inspirée de la méthode DATARUN.

Ces modèles sont incontournables pour aborder la phase suivante : la phase de conception et de réalisation de l'application. La prochaine section décrit une approche méthodique pour finaliser cette phase. Elle s'appuie sur un planning de réalisation découlant des cas d'utilisation.

LA PHASE DE CONCEPTION ET DE RÉALISATION

Traditionnellement, on fait une distinction assez nette entre la conception et la réalisation d'une application informatique. La conception consiste à traduire les besoins en spécifiant *comment* l'application pourra les satisfaire avant de procéder à sa réalisation avec des outils de développement appropriés : SGBD et langage de programmation essentiellement.

La conception précède toujours la réalisation, mais il n'est pas nécessaire que les spécifications de l'application, le *comment*, soient toutes élaborées avant de débuter la réalisation. Les méthodes de développement modernes s'appuient sur ce constat : dès qu'un nombre suffisant de spécifications est connu, il est souhaitable de les réaliser le plus tôt possible de manière à progresser efficacement dans le projet en menant en parallèle les tâches de conception et de réalisation. C'est la philosophie de développement que nous soutenons dans cet ouvrage.

Cette philosophie possède un corollaire : il est nécessaire d'avoir une vue d'ensemble du projet de développement avant d'aborder la conception et la réalisation car cette façon de faire exige une planification serrée. En effet, la réalisation de certaines spécifications n'est possible que si d'autres spécifications ont été au préalable réalisées. Il est donc essentiel d'échelonner dans le temps, selon une chronologie préalablement établie, la réalisation des diverses fonctionnalités de l'application.

Mener en parallèle conception et réalisation amène un bénéfice supplémentaire. Cela permet une validation immédiate de la faisabilité technique des spécifications. La réalisation ou la tentative de réalisation d'une spécification peut mener à des ajustements à la spécification, ajustements qui pourraient conditionner d'autres spécifications élaborées par la suite.

On donne aux méthodes de développement qui s'appuient sur cette philosophie le qualificatif de méthodes *progressives* de développement ou de méthodes de développement *incrémentiel*. Pour faire image, la métaphore suivante permet d'illustrer la méthode : l'application est réalisée en s'inspirant de la croissance de l'oignon, soit par ajout de couches successives, jusqu'à ce qu'il atteigne sa maturité.

Les étapes de la phase de conception et de réalisation

En vertu de la philosophie de développement évoquée plus haut, la phase de conception et de réalisation devrait comporter les étapes suivantes :

1. Traduire le modèle conceptuel de données produit à la phase d'analyse des besoins en un modèle relationnel de données en respectant les règles de dérivation vues au chapitre 2 ;
2. À partir du diagramme de cas d'utilisation, établir un planning de conception et de réalisation de chaque cas ; certains cas auront priorité sur d'autres ;
3. Chaque cas sera réalisé selon le planning en procédant de la manière suivante :
   a. Dessiner les panoramas requis pour supporter le cas si aucun prototype visuel n'a été produit durant la phase d'analyse des besoins ;
   b. Dessiner les rapports ou les états découlant du cas ;
   c. Mettre à jour le modèle relationnel pour tenir compte de données persistantes identifiées lors du dessin des panoramas et des états ;
   d. Réaliser le cas d'utilisation en créant d'abord le modèle physique, soit le schéma relationnel comportant toutes les contraintes d'intégrité sémantique et référentielle sur les tables nécessaires pour répondre aux spécifications des panoramas et des états ;
   e. Réaliser toutes les requêtes nécessaires pour produire les panoramas et les états ;
   f. Réaliser chaque panorama sur la base des requêtes élaborées plus tôt et des règles de gestion tirées du cas d'utilisation ;
   g. Réaliser chaque état sur la base des requêtes élaborées plus tôt.

La conception et la réalisation seront grandement accélérées si le modèle de fonctionnement de l'application a été produit au cours de la phase d'analyse. Comme en font foi les étapes de la méthode de développement proposée ci-dessus, le prototype visuel en est un élément essentiel. Si le prototype visuel existe déjà, l'étape 3.a peut être court-circuitée et la méthode devrait nous mener assez rapidement à la réalisation du modèle physique.

Nous allons reprendre l'étude de cas Club d'athlétisme (cas 4-8) au point où nous l'avions laissée, afin d'illustrer la démarche de conception et de réalisation dans le contexte où un prototype visuel existe déjà. Cette étude de cas ne vise pas l'exhaustivité au plan de l'illustration. Il est hors de portée de cet ouvrage de montrer comment, à l'aide d'un SGBD comme MS Access et du langage de programmation VBA sous-jacent, serait réalisé un panorama en procédant à la programmation des règles de gestion de l'application. Notre objectif est plutôt de faire ressortir la nature progressive de la méthode et de mettre en relief l'importance d'un planning judicieux de réalisation des cas d'utilisation.

CAS 4-9 CONCEPTION ET RÉALISATION DE L'APPLICATION CLUB D'ATHLÉTISME

Considérant le **Modèle de fonctionnement de l'application** qui a donné lieu au **Modèle conceptuel de données** produit au cours de l'étude de cas 4-8, proposer un planning de réalisation de l'application, traduire le modèle conceptuel en un modèle relationnel, réaliser le modèle physique pour développer le premier cas d'utilisation prévu au planning.

Le diagramme de cas d'utilisation pour le projet Club d'athlétisme produit dans le cadre de l'étude de cas 4-5 comporte cinq cas :

1. Inscription des membres
2. Compilation des recettes et des dépenses
3. Inscription aux compétitions
4. Organisation des activités
5. Compilation des résultats

Le planning de réalisation des cas repose sur un principe fort simple : il faut réaliser en premier lieu les cas dont les panoramas permettent la saisie de données qui devront être disponibles pour exploiter les panoramas d'autres cas d'utilisation. Par exemple, le cas **Organisation des activités** décrit le processus de saisie des données relatives à une activité. Le cas indique que les membres du comité d'organisation sont des membres actifs du club. On en déduit que l'on doit avoir procédé à l'inscription d'un membre avant que l'on puisse en faire un membre du comité d'organisation d'une activité. Le cas **Inscription des membres** est donc fonctionnellement préalable à **Organisation des activités**.

Il en va de même pour **Inscription aux compétitions** qui a comme préalable le cas **Organisation des activités**. De plus, le cas **Compilation des résultats** repose sur l'exigence que l'athlète soit au préalable inscrit à une compétition, dans le cas d'un résultat obtenu en compétition.

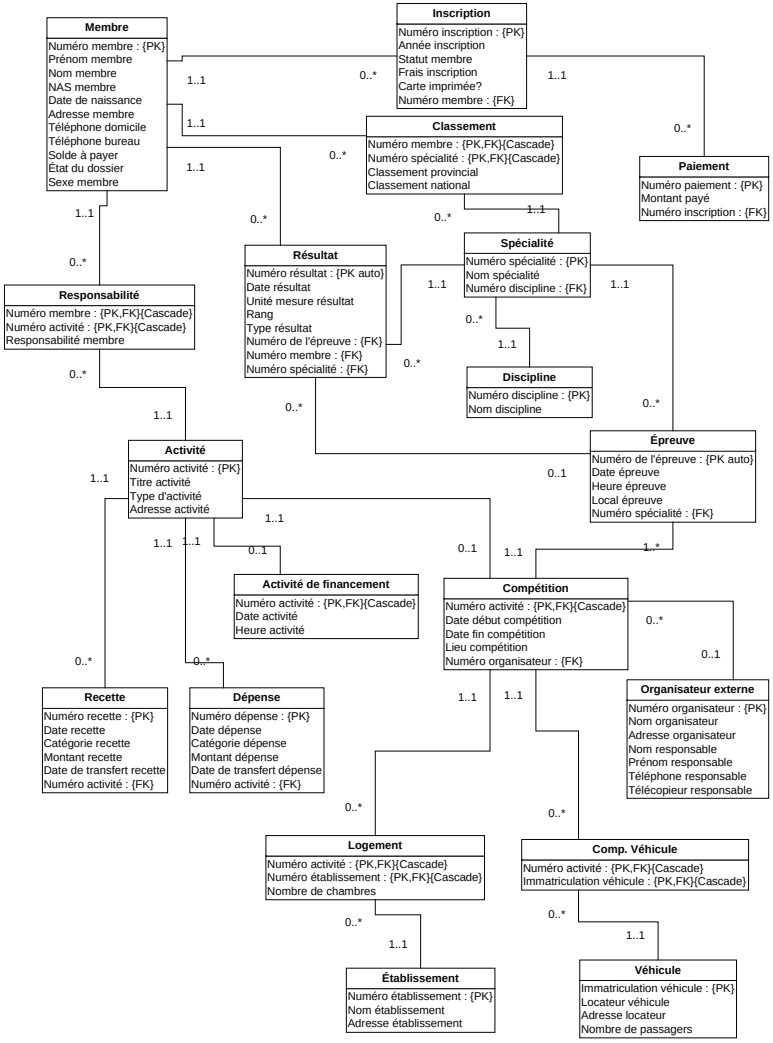
En vertu de ce principe, le planning coule de source :

1. Inscription des membres EST PRÉALABLE À
2. Organisation des activités EST PRÉALABLE À
3. Inscription aux compétitions EST PRÉALABLE À
4. Compilation des résultats
5. Compilation des recettes et des dépenses

Avant de réaliser le cas **Inscription des membres**, le modélisateur devra traduire le modèle conceptuel en un modèle relationnel de données.

Modèle relationnel dérivé du modèle conceptuel

Le passage du modèle conceptuel au modèle relationnel ne pose pas de réelle difficulté dans ce cas. On n'y retrouve que des associations binaires pouvant comporter une entité d'association à l'occasion. La principale difficulté concerne les associations spécialisées. On note la présence de deux associations d'héritage et d'une association de composition dans ce modèle conceptuel. Ces associations seront traitées en priorité.



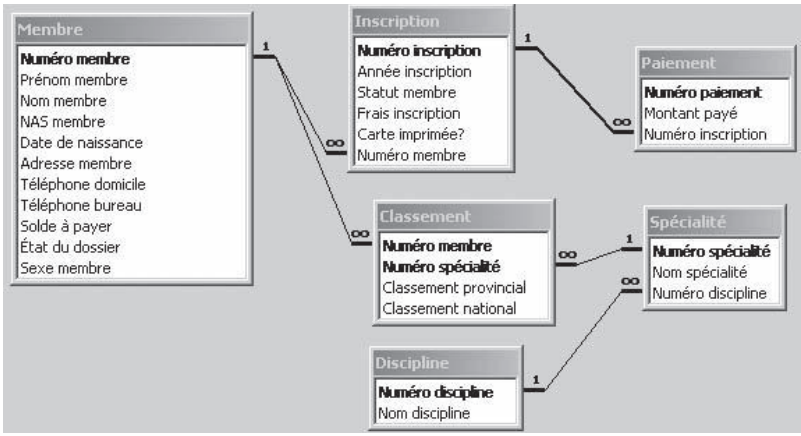
Les deux associations d'héritage montre une contrainte d'exclusion. En vertu des règles de dérivation, la table dérivée du supertype **Activité** doit comporter un champ permettant de distinguer le type d'activité. Or il y a déjà un attribut dans **Activité** qui joue ce rôle : **Type d'activité**. Par ailleurs, les tables dérivées des sous-types, **Activité de financement** et **Compétition**, auront comme clé primaire la clé primaire de leur table supertype : **Numéro activité**.

La clé primaire de la table **Compétition**, étant connue, la table dérivée de l'entité composant **Épreuve** devrait se voir attribuer une clé primaire composée de la clé primaire du composite **Numéro activité** et de son propre identifiant, soit **Numéro de l'épreuve**. Puisque la table **Résultat** est associée à **Épreuve** sur une base mère-fille, on doit retrouver la clé primaire de **Épreuve** comme clé étrangère dans **Résultat**. Pour éviter de voir une clé étrangère composée dans **Résultat**, la modélisateur a optimisé la clé primaire composée de **Épreuve** pour en faire une clé primaire simple à génération automatique de valeurs, d'où la présence de la mention {PK auto} sur le champ **Numéro de l'épreuve**.

Le modèle relationnel étant connu, il est possible d'aborder la conception et la réalisation du cas d'utilisation **Inscription des membres**. Les scénarios du cas font état des modalités d'inscription du membre, du paiement des frais et, dans le cas d'un athlète, du choix éventuel d'une spécialité nouvelle. Les panoramas en cause sont **Dossier membre** et **Dossier athlète**. Les onglets relatifs aux résultats de l'athlète sur **Dossier athlète** ne sont pas exploités dans ce cas d'utilisation.

À partir du modèle logique, des panoramas et des scénarios du cas d'utilisation **Inscription des membres**, on peut facilement établir quelles sont les tables qui devront être réalisées initialement dans le modèle physique. Il s'agit des tables **Membre**, **Inscription**, **Paiement**, **Classement**, **Spécialité** et **Discipline**. Elles comportent les données persistantes nécessaires à la réalisation des deux panoramas, excluant les onglets des résultats du panorama **Dossier athlète** qui seront pris en charge ultérieurement dans le planning de réalisation.

En vertu des règles de passage du modèle relationnel au modèle physique décrites au chapitre 3, le schéma relationnel sera ensuite réalisé. Nous proposons ci-après une illustration du schéma réalisée en Microsoft Access. L'illustration ne fait pas voir les contraintes d'intégrité sémantique et les contraintes de domaine sur les champs. Il va de soi que ces contraintes doivent être réalisées dans le modèle physique par le biais de propriétés spécifiques définies sur les champs. Par exemple, le champ **Carte imprimée ?** doit être de type booléen et sa valeur par défaut sera fixée à Non. Le champ **Statut membre** pourrait être de type entier avec pour valeurs acceptables (Valide si) 1 ou 2, correspondant respectivement à Actif et Inactif. **Frais inscription** pourrait avoir une valeur par défaut qui correspond aux frais d'inscription payés par les athlètes.


Membre
Numéro membre
Prénom membre
Nom membre
NAS membre
Date de naissance
Adresse membre
Téléphone domicile
Téléphone bureau
Solde à payer
État du dossier
Sexe membre
Inscription
Numéro inscription
Année inscription
Statut membre
Frais inscription
Carte imprimée?
Numéro membre
Paiement
Numéro paiement
Montant payé
Numéro inscription
Classement
Numéro membre
Numéro spécialité
Classement provincial
Classement national
Spécialité
Numéro spécialité
Nom spécialité
Numéro discipline
Discipline
Numéro discipline
Nom discipline
1
∞

Nous avons exposé au cours de ce chapitre une méthode d'analyse, de conception et de réalisation d'une application de base de données. La méthode découpe un projet de développement d'une application de base de données en deux phases :

1. L'analyse des besoins
2. La conception et la réalisation de l'application

La phase d'analyse donne lieu à la réalisation d'un livrable qui sera soit un modèle de fonctionnement du système d'information, soit un modèle de fonctionnement de l'application, selon le contexte dans lequel s'inscrit le projet. Quel que soit le modèle élaboré au cours de cette phase, deux éléments essentiels en ressortent : un modèle conceptuel de données et un modèle de cas d'utilisation. Ces deux éléments serviront à alimenter la deuxième phase de la méthode.

La phase de conception et de réalisation vise la production de l'application de base de données. Elle s'appuie sur une démarche systématique s'inspirant d'une philosophie de développement progressif et itératif de l'application. Pour ce faire, la démarche consiste à dériver du modèle conceptuel un modèle relationnel de données et à proposer un planning de développement qui dresse une chronologie de réalisation des cas d'utilisation, établie à l'aide d'un prototype visuel de l'application.

Bien qu'il n'existe aucun logiciel qui puisse supporter intégralement toutes les phases de la méthode présentée au cours de ce chapitre, un certain nombre d'outils informatiques peuvent être utilisés pour automatiser certains processus, notamment la dérivation d'un modèle relationnel à partir d'un modèle conceptuel de données ou la dérivation d'un modèle physique à partir d'un modèle relationnel de données.

Le prochain chapitre a pour objet de présenter et de comparer deux outils susceptibles de faciliter la tâche du modélisateur en lui permettant d'accélérer la réalisation des modèles de données nécessaires à chaque phase de la méthode.

EXERCICES D'ANALYSE DE BESOINS ET DE CONCEPTION D'UNE APPLICATION DE BASE DE DONNÉES

EXERCICE 4-1

Considérant la nature des opérations de l'Hôtel Rive-Sud présentée plus bas, élaborer un modèle de fonctionnement du système d'information de cette organisation comportant obligatoirement :

- Un descriptif de chaque document
- Un diagramme des cas d'utilisation
- Les scénarios de chaque cas d'utilisation
- Un recensement des données persistantes
- Un modèle conceptuel de données illustrant les données persistantes

Description des opérations

Hôtel Rive-Sud est un petit hôtel qui accueille une clientèle touristique ou d'affaires. Aucun système informatisé n'est utilisé à des fins de gestion. Toutes les activités de gestion sont de nature administrative et dans ce contexte, plusieurs documents de saisie sont exploités sous forme papier. Un exemplaire de chaque document est donné ci-après.

Les clients font leur réservation sous diverses formes : téléphone, télécopieur, courriel. Dans tous les cas, l'employé préposé à la réservation doit consigner sur la fiche « Réservation » toutes les informations requises dont la date de réservation, le numéro de réservation, le numéro de l'employé qui prend la réservation, le titre, le nom et le prénom du client, le nom de sa compagnie s'il y a lieu, et au moins un des éléments suivants : courriel, téléphone, télécopieur. Ensuite la date d'arrivée du client et le nombre de nuitées doivent être saisis. Selon le type de chambre exigé par le client (standard, de luxe ou suite) et l'occupation (simple, double, quadruple), l'employé inscrit le tarif de base donné par la grille tarifaire et, si le client appartient à un organisme bénéficiant d'un escompte, il inscrit de plus l'escompte accordé selon la table des escomptes négociés. L'employé calcule ensuite le tarif réellement accordé au client. Si le client exige une réservation garantie, le type de carte de crédit et le numéro de la carte doivent être inscrits sur la fiche.

Pour simplifier le problème, une réservation ne concerne qu'une seule chambre. Donc, si plusieurs chambres sont réservées, chacune fait l'objet d'une réservation différente avec sa propre fiche de réservation. De plus, même si un client se présente sans réservation, on remplit une fiche de réservation avant de lui faire compléter une fiche d'inscription.

Un journal quotidien des réservations, « Réservations du jour », permet de consigner jour après jour, selon les réservations reçues, les chambres allouées aux clients et le numéro de réservation correspondant.

Lorsque le client se présente pour prendre sa chambre, on a déjà dactylographié une fiche d'inscription qui lui est remise et où il doit inscrire son adresse, fournir les informations détaillées de sa carte de crédit et finalement signer.

Le titulaire de la carte peut être différent du client. Cependant, si le client paie avec une carte de crédit et qu'il s'agit d'une réservation garantie, la carte pour garantir la réservation sera la même que celle utilisée pour le paiement. Une et une seule carte de crédit est associée en tout temps à un client.

Pour chaque employé engagé, on a complété une fiche appelée « Fiche employé ». Toutes ces données sont essentielles et doivent être fournies en partie par l'employé pour son engagement. Seulement trois catégories de postes sont en vigueur : commis, préposé aux chambres, préposé à l'entretien.

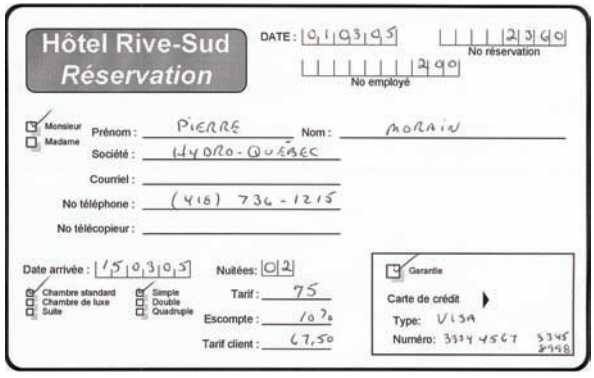
Chaque jour, tôt le matin, une feuille « Affectation des préposés aux chambres » est remplie permettant de déterminer, si une chambre a été occupée durant la nuit et quel employé sera chargé du ménage de la chambre.

L'hôtel fait appel à des fournisseurs attitrés pour des services spécialisés (plombier, menuisier, peintre, électricien). Des taux fixes sont négociés avec eux à chaque année. Ce taux et les données sur le fournisseur sont consignés sur une « Fiche fournisseur ».

Lorsqu'une réparation dans l'Hôtel exige une intervention d'un fournisseur attitré, une fiche « Appel de service » est complétée pour le service requis et le fournisseur retenu. On prend soin d'inscrire la raison de l'appel et dès que le travail est terminé, on inscrit la date et l'heure d'exécution (Fin des travaux). Si l'appel de service concerne une chambre en particulier, le numéro de la chambre est aussi inscrit sur la fiche.

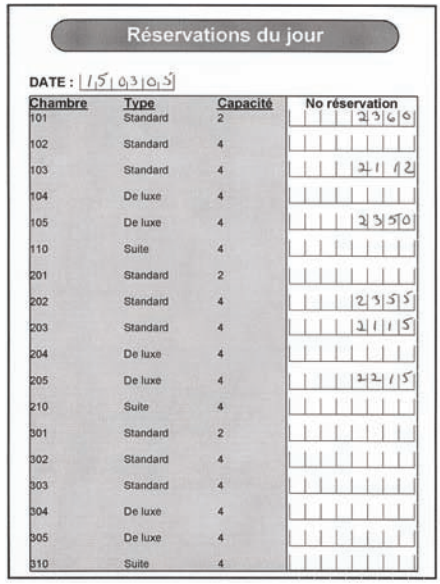
Une facture est produite pour le client à la fin de son séjour. Elle comporte, en plus des frais pour la chambre, les autres frais engagés lors de son séjour. Pour chaque frais la facture mentionne la date, le type de frais, le coût. Les types de frais se limitent à quatre : service aux chambres, minibar, interurbain et location de films. Le client doit acquitter le total de la facture incluant les taxes lors son départ. On inscrit alors la mention : Payé.

Des échantillons de chaque document utilisé dans le cadre de la gestion des opérations de l'hôtel sont fournis ci après.



Grille tarifaire	
Standard simple	75 $
Standard double	85 $
Standard quadruple	100 $
De luxe simple	95 $
De luxe double	120 $
De luxe quadruple	150 $
Suite simple	160 $
Suite double	160 $
Suite quadruple	185 $

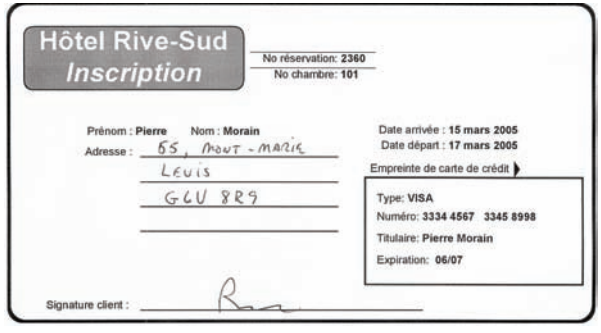
Escompte négocié	
Publique parapublique	10%
Ordre des ingénieurs	5%
Administrateurs agréés	7%
Corporation CA	7%
Corporation CGA	7%



Réservations du jour

DATE : 15/03/05

Chambre	Type	Capacité	No réservation
101	Standard	2	2360
102	Standard	4	
103	Standard	4	2412
104	De luxe	4	
105	De luxe	4	2350
110	Suite	4	
201	Standard	2	
202	Standard	4	2355
203	Standard	4	2115
204	De luxe	4	
205	De luxe	4	2215
210	Suite	4	
301	Standard	2	
302	Standard	4	
303	Standard	4	
304	De luxe	4	
305	De luxe	4	
310	Suite	4	



Hôtel Rive-Sud
Inscription

No réservation: **2360**
No chambre: **101**

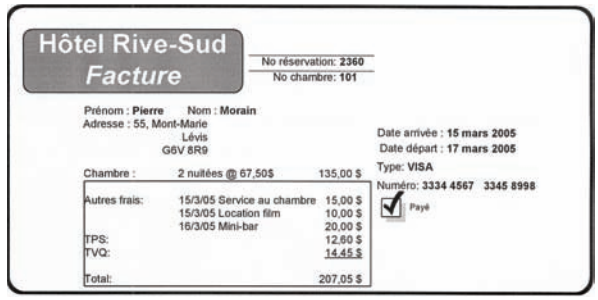
Prénom : **Pierre** Nom : **Morain**
Adresse : 55, Mont-Marie
Levis
G6V 8R9

Date arrivée : **15 mars 2005**
Date départ : **17 mars 2005**

Empreinte de carte de crédit

Type: **VISA**
Numéro: **3334 4567 3345 8998**
Titulaire: **Pierre Morain**
Expiration: **06/07**

Signature client :



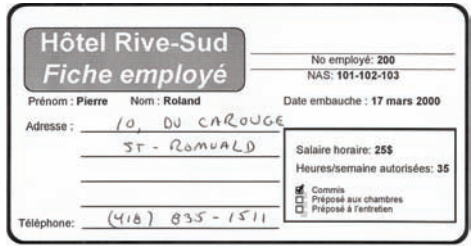
Hôtel Rive-Sud
Facture

No réservation: **2360**
No chambre: **101**

Prénom : **Pierre** Nom : **Morain**
Adresse : 55, Mont-Marie
Lévis
G6V 8R9

Date arrivée : **15 mars 2005**
Date départ : **17 mars 2005**
Type: **VISA**
Numéro: **3334 4567 3345 8998**
☑ Payé

Chambre :	2 nuitées @ 67,50$	135,00 $
Autres frais:	15/3/05 Service au chambre	15,00 $
	15/3/05 Location film	10,00 $
	16/3/05 Mini-bar	20,00 $
TPS:		12,60 $
TVQ:		14,45 $
Total:		207,05 $



Hôtel Rive-Sud
Fiche employé

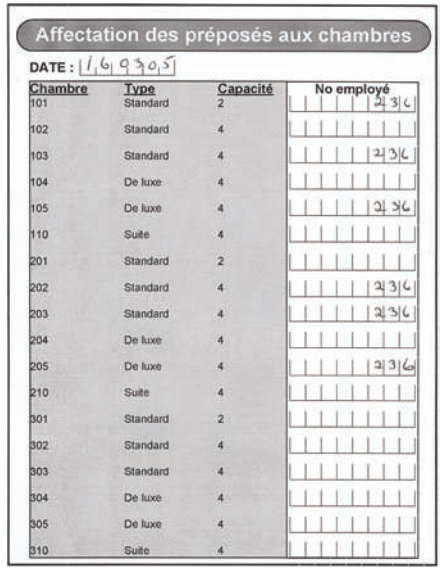
No employé: **200**
NAS: **101-102-103**

Prénom : **Pierre** Nom : **Roland** Date embauche : **17 mars 2000**

Adresse : 10, DU CAROUGE
ST-ROMUALD

Téléphone: (418) 835-1511

Salaire horaire: **25$**
Heures/semaine autorisées: **35**
☑ Commis
☐ Préposé aux chambres
☐ Préposé à l'entretien



Affectation des préposés aux chambres

DATE : 1,6,9,3,0,5

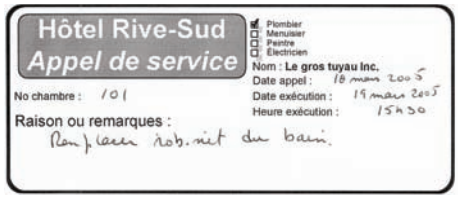
Chambre	Type	Capacité	No employé
101	Standard	2	236
102	Standard	4	
103	Standard	4	236
104	De luxe	4	
105	De luxe	4	236
110	Suite	4	
201	Standard	2	
202	Standard	4	236
203	Standard	4	236
204	De luxe	4	
205	De luxe	4	236
210	Suite	4	
301	Standard	2	
302	Standard	4	
303	Standard	4	
304	De luxe	4	
305	De luxe	4	
310	Suite	4	

Hôtel Rive-Sud
Fiche fournisseur

☑ Plombier
☐ Menuisier
☐ Peintre
☐ Électricien

No fournisseur : **36**
Nom : **Le gros tuyau Inc.**
Tarif horaire négocié: **55$/h.**

Télécopieur : **418-833-6764**
Téléphone service : **418-833-3434**



EXERCICE 4-2

La direction de l'Hôtel Rive-sud souhaite procéder à la réalisation d'une application de base de données qui permettra d'informatiser les cas d'utilisation du système d'information. Débuter cette deuxième phase en dérivant un modèle relationnel à partir du modèle conceptuel de données élaboré dans le cadre de l'exercice 4-1.

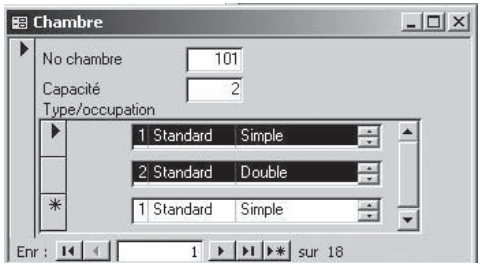
Considérant le prototype visuel dont les principaux panoramas sont donnés ci-après, rédiger ensuite les scénarios de cas d'utilisation prescrivant les modalités d'interactions des utilisateurs avec les panoramas. Identifier à l'aide de ces scénarios les données persistantes non recensées dans le modèle de fonctionnement du système d'information. Mettre à jour le modèle relationnel pour tenir compte de ces nouvelles données.

Proposer un planning de réalisation des cas d'utilisation.

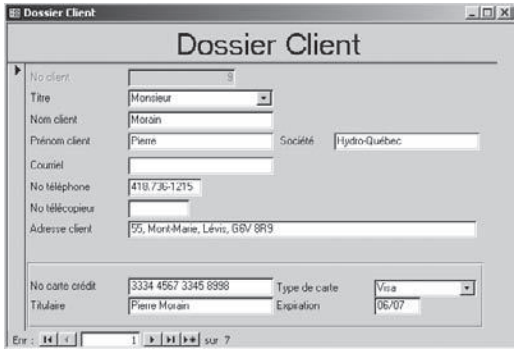
Dériver le modèle physique à partir du modèle relationnel de données en réalisant le schéma relationnel en MS Access.

Panoramas du prototype visuel.

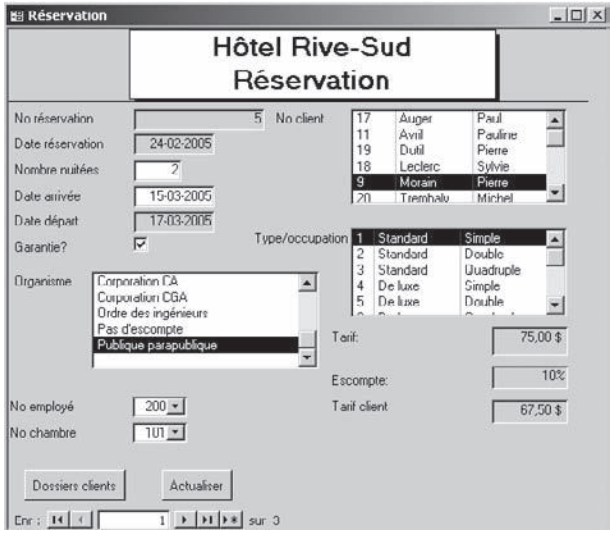
Un panorama pour la saisie et la consultation des données sur les chambres. Le dessin de ce formulaire est donné ci-dessous, il porte le nom « Chambre ». Il permet d'associer à une chambre un type et les formes d'occupation possibles.



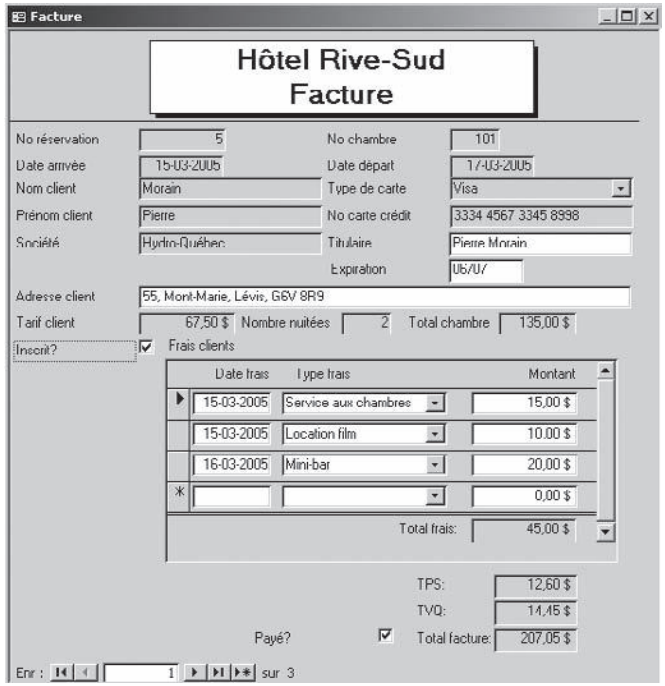
Un panorama pour faire la saisie et la consultation des données conservées sur un client. Le dessin porte le nom « Dossier client ».

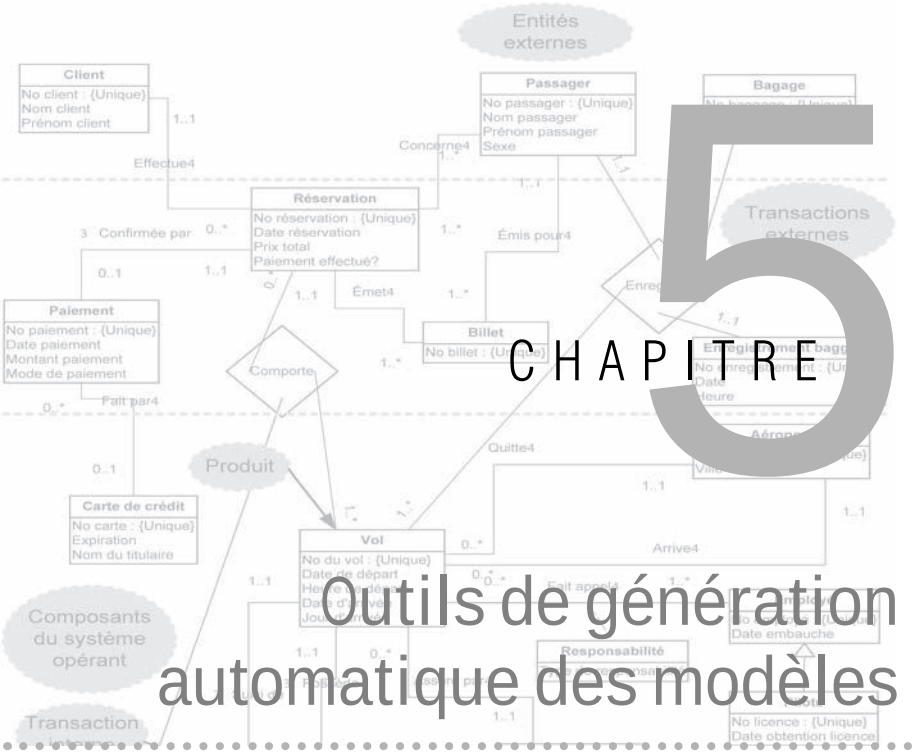


Un panorama permettant de saisir et de consulter les réservations. Le dessin de ce panorama porte le nom « Réservation ». Si la réservation concerne un nouveau client, un bouton commande donne accès au formulaire « Dossier client » présenté plus tôt. Cela va permettre de saisir les données sur le client avant de revenir au formulaire « Réservation » pour saisir la réservation.



Un panorama pour faire la saisie et la consultation des données complétées lors de l'inscription du client ainsi que des frais encourus lors du séjour. Le même panorama appelé « Facture » permet de confirmer le paiement et d'imprimer la facture.





CHAPITRE 5

Outils de génération automatique des modèles

OBJECTIFS

- Comment les outils informatiques permettent au modélisateur d'effectuer la dérivation d'un modèle à partir d'un autre.
- Quelles sont les limites de tels outils.
- Caractéristiques de deux outils supportant la notation UML et permettant de réaliser des modèles de données et des modèles de cas d'utilisation.

Nous n'avons pas la prétention de faire ici une évaluation exhaustive des outils disponibles sur le marché pour réaliser des modèles de données. On pourrait consacrer à cet exercice un ouvrage complet. Notre choix s'est arrêté sur deux outils qui répondent aux critères suivants :

1. l'outil permet de réaliser des modèles de données aux niveaux conceptuel, logique et physique ;
2. l'outil automatise la dérivation d'un modèle logique à partir d'un modèle conceptuel et vice versa ;

3. l'outil automatise la dérivation d'un modèle physique à partir d'un modèle logique et vice versa ;
4. l'outil permet de réaliser les modèles de niveau conceptuel et de niveau logique à l'aide de la notation UML ;
5. l'outil produit un modèle physique par génération de script SQL pour une large panoplie de SGBD ;
6. l'outil permet de réaliser un modèle de cas d'utilisation à l'aide de la notation UML.

Les outils de modélisation qui seront étudiés sont PowerAMC de Sybase, un éditeur de SGBD, et Win'Design de l'éditeur français de logiciels CECIMA. Ces produits, ayant atteint un niveau de maturité avancée, répondent aux critères énoncés plus haut et sont maintenant largement utilisés dans l'industrie.

PowerAMC et Win'Design sont disponibles en langue française.

POWERAMC

Bien que PowerAMC permette de réaliser un modèle conceptuel de données (MCD), ce type de modèle ne peut être réalisé qu'avec la notation Merise ou la notation de Chen (multiplicités représentées en *ergot de coq*). Afin de réaliser un modèle conceptuel de données avec la notation UML, le modélisateur doit réaliser un *modèle orienté objet* (MOO) sous forme d'un diagramme de classes. Ceci peut porter à confusion mais ne représente pas une contrainte majeure. Comme nous l'avons vu au chapitre 1, un modèle conceptuel de données s'exprime en UML sous forme d'un diagramme de classes où chaque classe possède le stéréotype «*entité*» et ne comporte que des attributs.

Dès lors qu'un modèle de type MOO a été réalisé, un modèle logique de données pourra être dérivé automatiquement de ce dernier. Là encore, la terminologie utilisée dans PowerAMC peut porter à confusion car ce qui est dérivé d'un MOO est appelé une MPD (Modèle physique de données). Or, un MPD dans PowerAMC n'est rien d'autre qu'un schéma relationnel d'une BD, ce qui correspond en fait au modèle relationnel tel que défini au chapitre 2. Le véritable modèle physique est le script SQL, dérivé du MPD, que PowerAMC génère pour réaliser une BD.

En somme, PowerAMC peut produire des modèles de données de trois niveaux : conceptuel, logique et physique, mais la terminologie utilisée introduit une confusion des genres. Ce qui n'enlève rien aux possibilités de PowerAMC car il répond en tout point aux critères énumérés plus haut.

Nous allons considérer à tour de rôle les modalités de création des modèles conceptuel, logique et physique avec PowerAMC version 11.0 et les conditions à respecter pour assurer la dérivation automatique d'un modèle à partir d'un autre. Avant de s'engager dans cette étude, il y a lieu de s'attarder au choix de certaines options du logiciel ayant pour objet de faciliter la gestion des modèles.

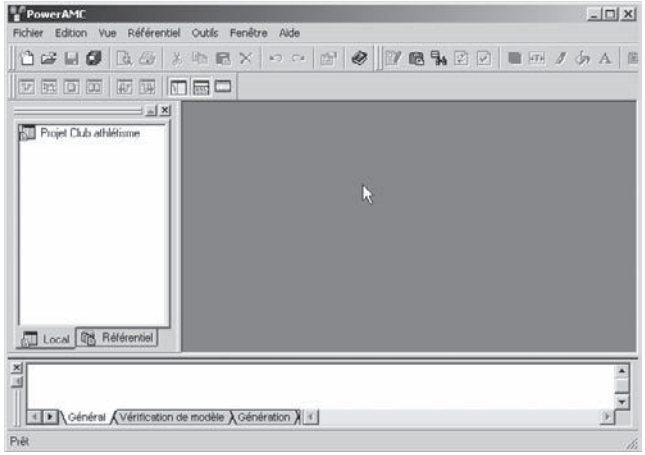
Démarrage et fixation des paramètres de PowerAMC

Tous les modèles réalisés dans le cadre du même projet sont conservés dans un *Espace de travail*. La première fois que le logiciel est lancé, un espace de travail vide est présenté portant le nom *Espace de travail*. Ce nom peut être changé en cliquant deux fois sur l'étiquette du nom. La figure 5-1 montre un espace de travail vide portant le nom *Projet Club athlétisme*.

Il est fortement suggéré de sauvegarder l'espace de travail initial, bien que vide, de manière à créer dans un répertoire approprié le ficher de type *.sws* qui comporte les références à tous les modèles du projet :

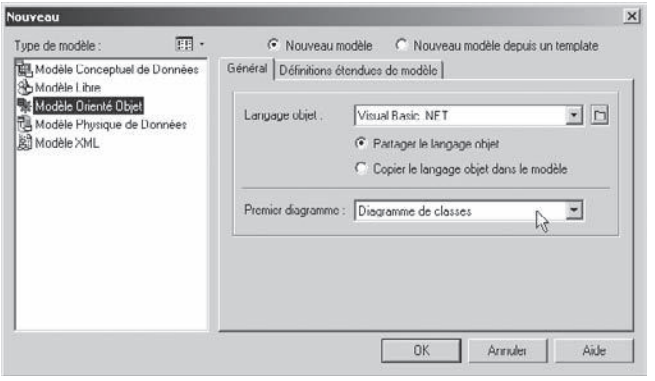
`Fichier->Enregistrer l'espace de travail sous…`

FIGURE 5-1 **Fenêtre de démarrage de PowerAMC**



`Fichier->Nouveau…`
ouvre un panorama permettant d'ajouter un modèle à l'espace de travail. La figure 5-2 montre les options qui s'offrent à l'utilisateur lorsqu'un Modèle Orienté Objet (MOO) doit être réalisé.

FIGURE 5-2 **Création d'un nouveau modèle dans l'espace de travail**



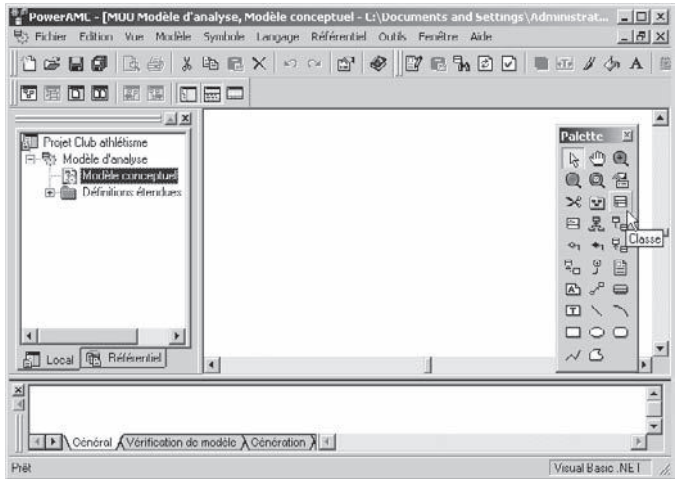
Création d'un modèle conceptuel de données avec PowerAMC

La réalisation d'un modèle conceptuel en UML doit impérativement se faire avec un MOO et non pas avec un modèle de type MCD (Modèle conceptuel de données avec les formalismes Merise ou Chen). Nous suggérons de sélectionner le langage objet *Visual Basic.NET* et obligatoirement *Diagramme de classes* à titre de premier diagramme, comme le montre la figure 5-2.

PowerAMC crée un modèle appelé par défaut **ModèleOrienteObjet_1**, dont le nom devrait être changé pour mieux illustrer la nature des documents créés dans le modèle. Nous suggérons **Modèle d'analyse** car il devrait permettre de regrouper des modèles produits dans la phase d'analyse notamment le modèle conceptuel de données et le diagramme de cas d'utilisation. Le modèle d'analyse comportera au départ **DiagrammeClasses_1** dont le nom devrait être changé pour mieux refléter sa nature soit **Modèle conceptuel de données**.

La figure 5-3 montre l'état initial de l'espace de travail après changement de noms des modèles.

FIGURE 5-3 **Création du Modèle conceptuel dans l'espace de travail**



Le panneau de droite va permettre de créer une représentation graphique des entités et des associations à l'aide d'une palette d'outils. Le tableau 5-1 reprend certains éléments de la palette qui sont pertinents à la réalisation d'un modèle conceptuel de données.

TABLEAU 5-1 **Modèle conceptuel et outils pertinents**

Icône	Terminologie UML	Dénomination au niveau conceptuel
	Classe	Entité
	Association	Association binaire
	Généralisation	Association d'héritage
	Composition	Association de composition

PowerAMC Version 11 pour Windows, dont certaines icônes de sa palette d'outils sont présentées à la figure 5-3, impose les limites suivantes quant à la réalisation d'un modèle conceptuel de données :

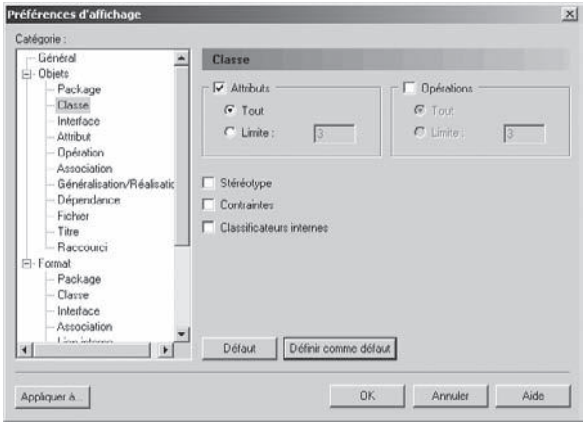
- une association de degré supérieur à 2 ne peut être exprimée, ce qui implique que tout modèle conceptuel doit être simplifié au point de ne comporter que des associations binaires. S'il subsiste des associations de degré supérieur, elles doivent être transformées en entités selon une procédure décrite plus loin.

- aucune contrainte inter-association ne peut être exprimée ; en conséquence, le modèle logique de données dérivé d'un modèle conceptuel doit être modifié pour tenir compte de contraintes inter-association le cas échéant.
- les contraintes d'intégrité sémantique que l'on peut formuler dans le modèle conceptuel sont les suivantes : liste de valeurs acceptables, valeur minimum, valeur maximum, valeur par défaut ; les autres contraintes sont formulées textuellement et ne sont pas prises en charge lors du passage au modèle logique.
- le type de données *Monnaie* peut être utilisé dans un modèle conceptuel mais sous le type *MN* .

Avant de procéder à la réalisation d'un premier modèle conceptuel de données, il importe de fixer des paramètres qui seront appliqués par défaut à tout modèle de ce genre. Ces paramètres concernent les préférences d'affichage du modèle : il n'y a pas lieu, par exemple, d'afficher les opérations d'une classe et il est par ailleurs nécessaire de faire voir le nom de chaque association.

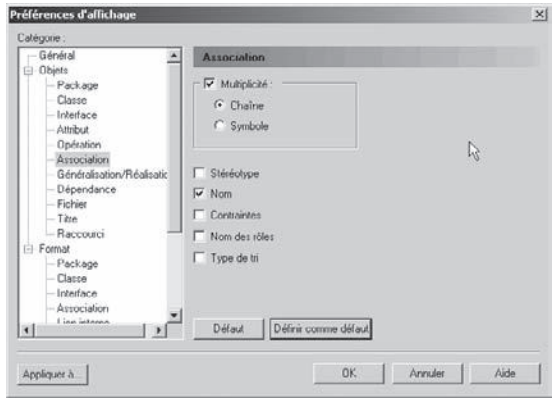
`Outils->Préférence d'affichage…`
permet de régler l'affichage d'une classe (entité) sans les opérations selon les paramètres donnés à la figure 5-4. En faire un choix par défaut en cliquant sur le bouton *Définir comme défaut*.

FIGURE 5-4 **Préférences d'affichage d'une classe**



Il en va de même pour l'affichage des associations réglé selon les préférences de la figure 5-5 et qui sera établi par défaut avec le bouton *Définir comme défaut*.

FIGURE 5-5 **Préférences d'affichage d'une association**



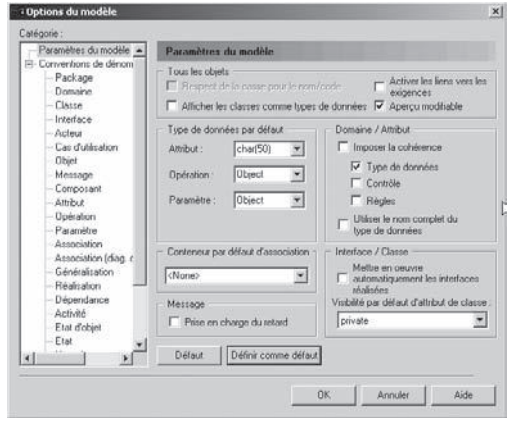
Le dernier paramètre qui doit être fixé par défaut est le type de données d'un attribut. Sachant que la majorité des attributs sont de type texte avec une longueur maximale de 50 caractères, le type proposé par défaut pourrait être char(50).

Pour ce faire, exécuter :

`Outils->Options du modèle…`

Ce paramètre doit être fixé en saisissant *char(50)* dans la zone **Attribut** de la fenêtre *Options du modèle* sans faire appel à la liste prédéfinie car le type *char* qui y est présent représente une chaîne de 255 caractères. Pour confirmer ce choix par défaut, cliquer sur le bouton *Définir comme défaut* placé au bas de la fenêtre (Figure 5-6).

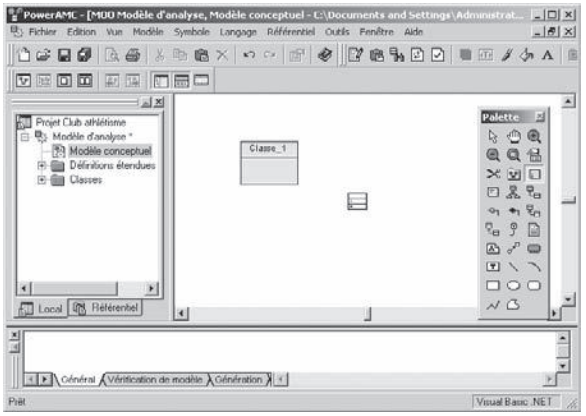
FIGURE 5-6 **Fixation du type de données par défaut**



Création d'une entité

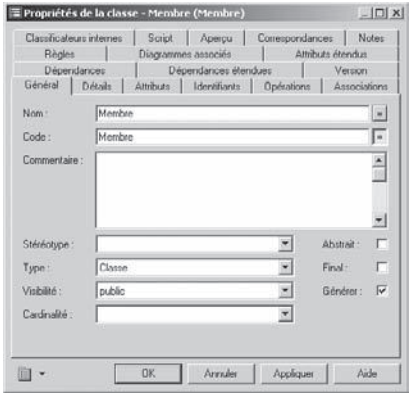
En vertu du tableau 5-1, une entité doit être créée avec l'outil classe . Il suffit de cliquer dans le panneau de droite pour y disposer une ou des entités dont les noms par défaut débutent par `Classe_` . Dès l'instant qu'une entité est créée, il suffit d'effectuer un double-clic sur celle-ci pour définir les propriétés de l'entité, notamment ses attributs, leur type de données, leurs contraintes d'intégrité et l'identifiant.

FIGURE 5-7 **Création d'une entité**



Le nom de l'entité est saisi dans la zone **Nom:** . Le nom codé de l'entité est automatiquement inscrit dans la zone **Code:**, l'utilisateur n'a pas à s'en soucier (Figure 5-8). **Générer** est coché par défaut pour assurer la génération d'une table pour cette entité dans le modèle logique.

FIGURE 5-8 **Propriétés de l'entité**

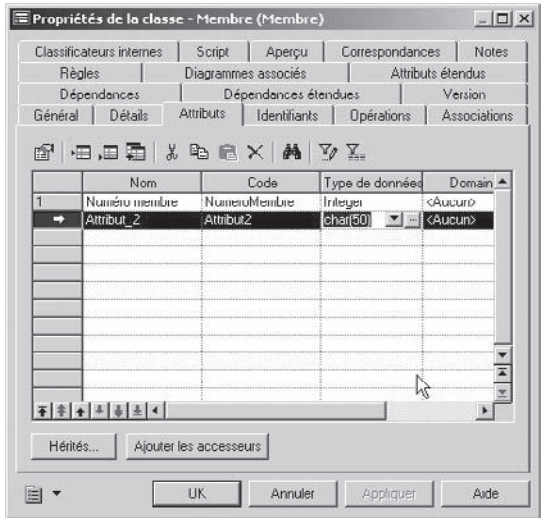


Sous l'onglet *Détails*, l'option **Persistant** doit être cochée. Il s'agit cependant d'une option définie par défaut. L'onglet *Attributs* doit être activé pour inscrire les attributs et choisir leur type. Leur nom codé est généré automatiquement. Le type par défaut *char(50)* peut être remplacé par un des types prédéfinis offerts dans la zone de liste déroulante **Type de données**. Les types de données de base exposés au chapitre 1 sont repris dans le tableau 5-2 et mis en correspondance avec les types PowerAMC offerts pour un modèle conceptuel. La figure 5-9 montre un attribut déjà saisi, **Numéro membre**, et un deuxième sur le point d'être saisi.

TABLEAU 5-2 **Types de données PowerAMC dans un modèle conceptuel**

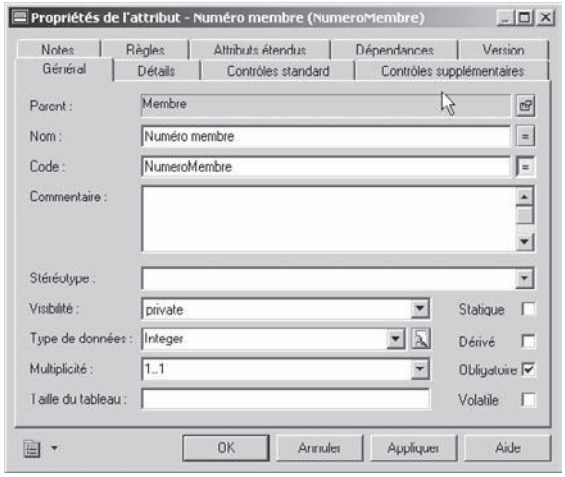
Type	Type PowerAMC
`Integer`	Integer
`Real`	Decimal
`String`	Char
`Date`	Date
`Boolean`	Boolean
`enum{}`	Le type doit être choisi parmi les types précédents et la liste des valeurs possibles sera donnée par l'onglet *Contrôles standard* de la fenêtre permettant de fixer les propriétés de l'attribut.

FIGURE 5-9 **Saisie des attributs de l'entité Membre et de leurs types de données**



Un double clic sur le numéro de l'attribut permet de saisir les autres propriétés de l'attribut, soit essentiellement les contraintes d'intégrité sémantique appliquées à l'attribut. La figure 5-10 fait voir les trois onglets de base permettant de définir les contraintes d'intégrité de l'attribut : *Général*, *Détails* et *Contrôles standard*.

FIGURE 5-10 **Saisie des propriétés de l'attribut**



Nous présentons dans le tableau 5-3 les contraintes d'intégrité sémantique que le modélisateur peut exprimer dans un modèle conceptuel réalisé avec PowerAMC et l'onglet approprié.

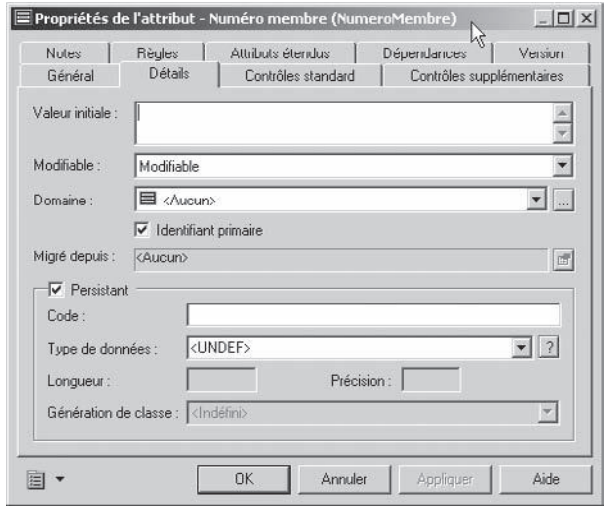
TABLEAU 5-3 **Contraintes d'intégrité sémantique dans un modèle conceptuel**

Contrainte	Onglet	Zones concernées
Non nul	Général	**Obligatoire** doit être coché (**Multiplicité** affiche automatiquement 1..1)
Unique	Détails	**Identifiant primaire** doit être coché
Valeur par défaut	Contrôles standard	**Défaut** doit être complété avec la valeur par défaut
Valeur minimum	Contrôles standard	**Minimum**
Valeur maximum	Contrôles standard	**Maximum**
Valeurs admissibles	Contrôles standard	**Liste des valeurs possibles**, colonne **Valeur,** les valeurs admissibles, colonne **Libellé**, l'étiquette associée à la valeur si celle-ci est codée

Notons à la figure 5-10 que l'attribut **Numéro membre** qui sera éventuellement défini comme identifiant, exige une valeur non nulle (**Obligatoire** est coché). La figure 5-11 montre les propriétés de l'attribut **Numéro membre** qui ont été définies sous l'onglet *Détails*. Comme il s'agit de l'identifiant de l'entité **Membre**, l'option **Identifiant primaire** est cochée.

Le nom de l'identifiant sera souligné dans le modèle conceptuel, ce qui correspond à la mention *{Unique}*. Si l'identifiant est composé, chaque attribut constituant l'identifiant sera souligné.

FIGURE 5-11 **Saisie des contraintes sous l'onglet Détails**



Comme le montre la figure 5-12, l'attribut **État du dossier** a été défini de type `Integer` et possède deux valeurs admissibles soit 0 ou 1; ce qui permet de coder les deux états possibles: Actif et Inactif. L'onglet *Contrôles standard* a été mis à contribution pour fixer les valeurs admissibles, la valeur minimum, la valeur maximum et la valeur par défaut de l'attribut **État du dossier**.

Si un attribut qui comporte une liste de valeurs admissibles est du type *Char*, les valeurs possibles ne peuvent comporter d'espace. Par exemple, l'attribut **Type activité** pourrait avoir comme valeur possible *Activité de financement*. Il serait plus approprié d'écrire *Activité-de-financement* comme valeur pour éviter une erreur lors de la génération du modèle logique.

FIGURE 5-12 **Saisie des contraintes sous l'onglet Contrôles standard**

Création d'une association binaire

La palette comporte l'outil *Association* qui, une fois sélectionné, permet de créer une association entre deux entités en cliquant sur l'une d'elles et en glissant le curseur vers la deuxième en conservant la pression sur le bouton gauche.

La figure 5-13 montre le résultat de la création d'une association binaire entre les entités **Membre** et **Activité**. Par défaut l'association porte un nom débutant par `Association_` et elle possède des multiplicités 0..1 et 0..* . Elle est représentée par une flèche, appelée *navigation*, ce qui n'est pas pertinent pour une association dans un modèle conceptuel de données.

Il suffit de faire un double clic sur l'association pour régler les propriétés de l'association permettant entre autres de retirer cette flèche. La fenêtre des propriétés de l'association comporte un onglet *Général* pour changer le nom de l'association dans la zone **Nom**:. La nom codé s'inscrit automatiquement dans la zone **Code**:. L'onglet *Détails* doit être activé pour ajuster les autres propriétés. Comme le décrit la figure 5-14, cet onglet permet d'effacer la flèche de navigation en retirant le crochet de l'option **Navigable**. Les multiplicités à chaque extrémité de l'association **Est responsable de** sont établies à l'aide de zones de liste déroulantes, disponibles sous le même onglet *Détails*.

FIGURE 5-13 **Propriétés données par défaut à une association**

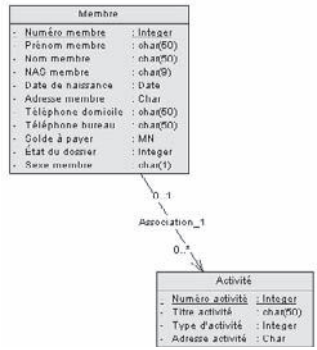
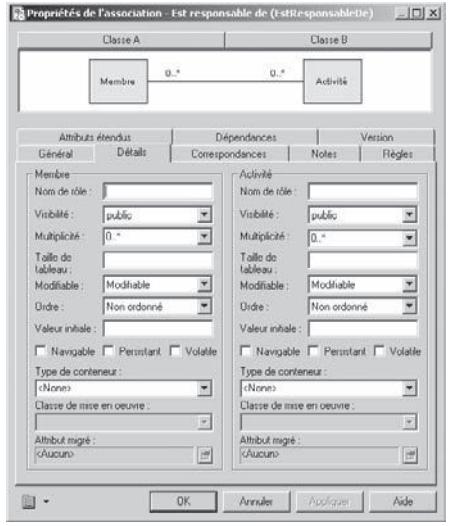
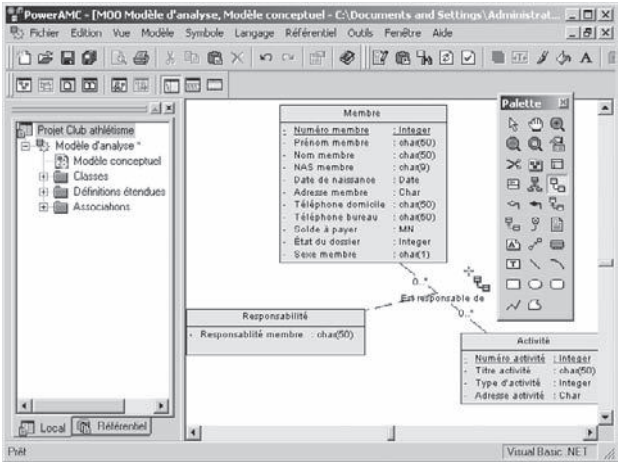


FIGURE 5-14 **Réglage des propriétés d'une association**



Lorsqu'une association binaire porte une entité d'association, l'association doit d'abord être créée comme décrit ci dessus. L'entité qui sera portée par l'association sera créée ensuite et en tout dernier lieu, avec l'outil *Association* . Le modélisateur doit cliquer sur l'entité que doit porter l'association et glisser le curseur vers cette association en conservant la pression sur le bouton gauche. En relâchant la pression sur le bouton gauche, une ligne pointillée s'inscrit entre l'entité et l'association. L'entité se transforme alors en entité d'association.

FIGURE 5-15 **Création d'une entité d'association Responsabilité**

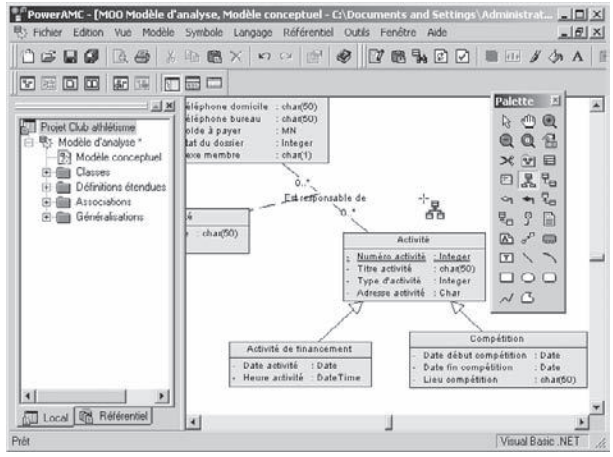


Création d'une association d'héritage ou de composition

L'association d'héritage se construit entre deux entités avec l'outil *Généralisation* . Il suffit de cliquer sur l'entité sous-type et de glisser le curseur vers l'entité supertype. Lorsque le bouton gauche est relâché, une association portant le triangle pointé vers le supertype apparaît.

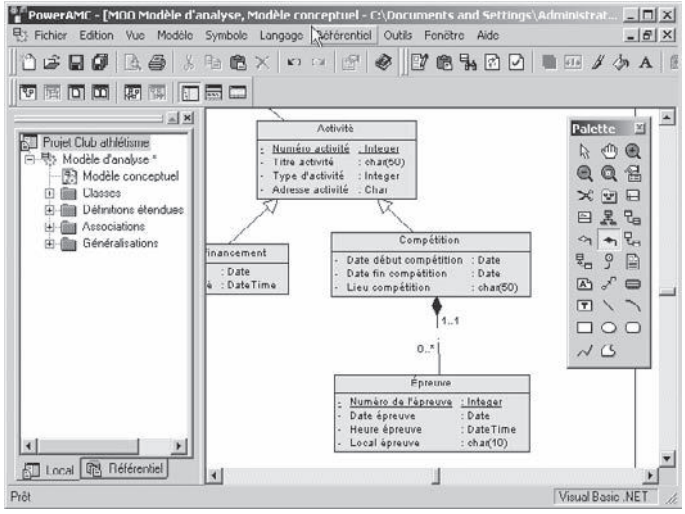
La figure 5-16 montre deux associations d'héritage créées sur la même entité supertype : **Activité**.

FIGURE 5-16 **Création de deux associations d'héritage**



On procède de manière similaire pour réaliser une association de composition avec l'outil *Composition* . Il suffit de cliquer sur l'entité composite et de glisser le curseur vers l'entité composant. Lorsque le bouton gauche est relâché, une association portant le losange noir pointé vers le composant s'affiche. Elle comporte un nom, une navigation et des multiplicités qui devront être ajustées à travers la fenêtre des propriétés de l'association. Nous suggérons de remplacer le nom par un point pour masquer le nom prédéfini et de donner une multiplicité 1..1 du côté composant. L'option **Navigable** doit être retirée du côté composant pour effacer la flèche. La figure 5-17 montre le résultat obtenu.

FIGURE 5-17 **Création d'une association de composition**



Création d'une association de degré supérieur

Comme nous le mentionnions au début de cette rubrique portant sur la création d'un modèle conceptuel de données avec PowerAMC, il n'est pas possible de créer une association de degré supérieur avec cet outil. Pour contourner cette incapacité, il faudra créer une entité éventuellement non dotée d'attributs pour représenter l'association et y rattacher, avec des associations de composition, toutes les entités impliquées dans l'association.

Considérons d'abord une entité de degré supérieur sans entité d'association comme le montre la figure 5-18. Une entité *composant* appelée **Comporte** sera créée sans attributs. Elle sera associée à **Vol**, **Réservation** et

Tarif par autant d'associations de composition où ces entités seront représentées comme *composites*. Les multiplicités du côté de **Comporte** seront toutes réglées à 0..*. Celles du côté des entités impliquées dans l'association de degré supérieur seront toutes 1..1.

PowerAMC pourra générer un modèle logique équivalent à celui dérivé du modèle conceptuel original si les règles théoriques de dérivation sont appliquées sur ce dernier. Par contre les multiplicités du modèle original ne peuvent être réalisées dans le nouveau modèle conceptuel.

Le modèle conceptuel, résultant des transformations au modèle 5-18 décrit ci-dessus, est illustré à la figure 5-19.

FIGURE 5-18 **Réservation auprès d'une société aérienne**

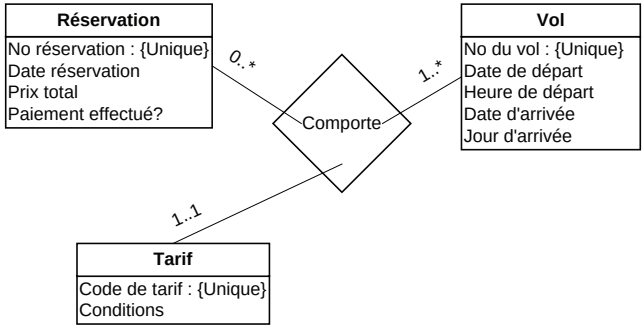
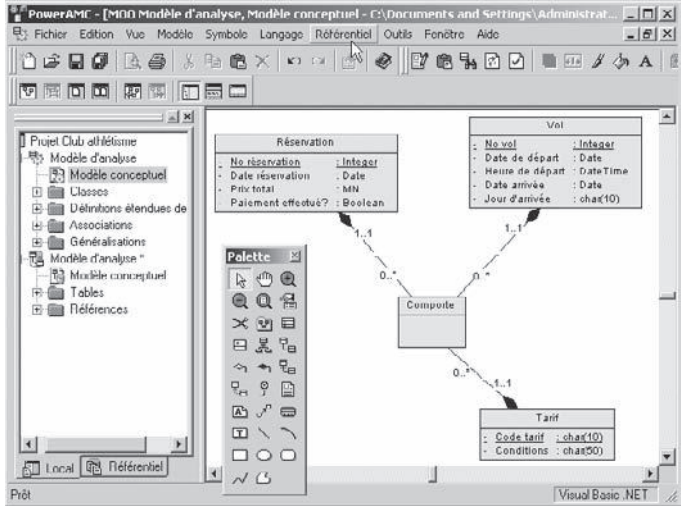


FIGURE 5-19 **Réservation auprès d'une société aérienne modélisée avec PowerAMC**



Considérons maintenant une entité de degré supérieur *avec* une entité d'association comme le propose la figure 5-20. L'entité d'association **Période de cours** devient une entité *composant*. Elle sera associée directement à **Professeur**, **Cours**, **Local** et **Jour semaine** par autant d'associations de composition où ces entités seront représentées comme *composites*. Les multiplicités du côté de **Période de cours** seront toutes réglées à 0..*. Celles du côté des entités impliquées dans l'association de degré supérieur seront toutes 1..1.

FIGURE 5-20 **Horaire des professeurs dans un établissement scolaire**

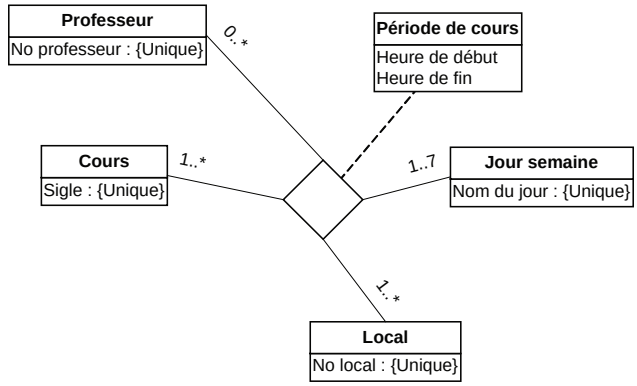
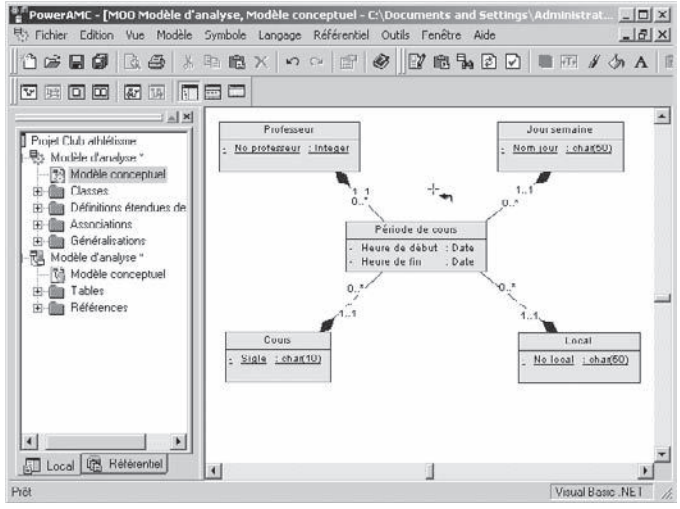


FIGURE 5-21 **Modèle 5-20 transformé pour représenter l'association de degré supérieur avec PowerAMC**



Génération d'un modèle logique de données avec PowerAMC

Si le modélisateur a pris soin de doter chaque attribut d'un type de données et de contraintes d'intégrité sémantique appropriées, il est possible de générer un modèle logique de données à partir d'un modèle conceptuel qui respectera la plupart des règles théoriques de dérivation.

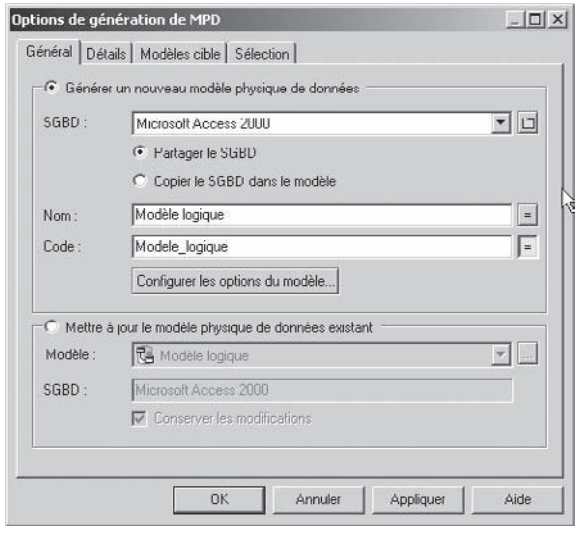
Le modélisateur doit s'assurer que chaque entité possède un identifiant explicite, sauf pour une entité d'association ou une entité agissant comme sous-type dans une association d'héritage ou agissant encore comme composant dans une association de composition. Dans ces trois derniers cas l'identifiant est implicite. On se souvient que la composition des clés primaires des tables dérivées dans le modèle logique repose sur la présence des identifiants explicites dans le modèle conceptuel.

Lorsqu'un modèle conceptuel est présent à l'écran, le modélisateur peut lancer la génération du modèle logique avec la commande :

`Outils->Générer un Modèle Physique de Données…`

Cette commande produit en fait un modèle logique qui sera lié à un SGBD en particulier, donc un modèle logique qui comporte certaines propriétés qui relèvent d'un modèle physique. Les choix relatifs à la génération du modèle logique sont faits à travers la fenêtre apparaissant à la figure 5-22.

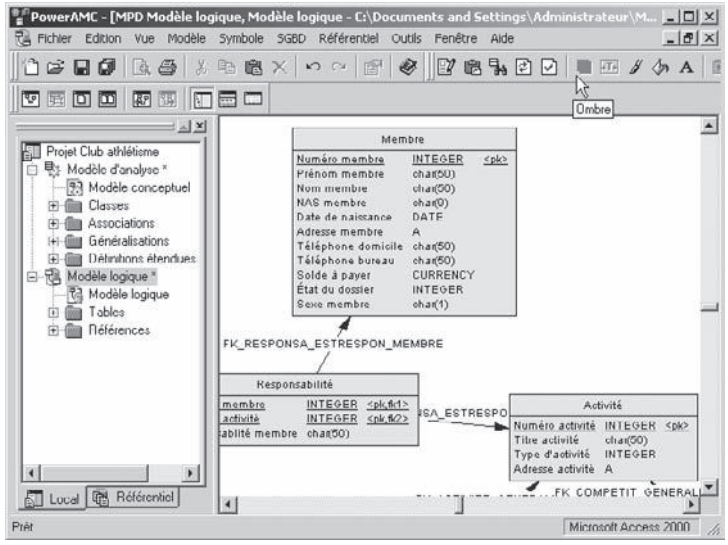
FIGURE 5-22 **Paramètres de génération des modèles logique et physique**



Lors de la première génération, le choix du SGBD doit être fait et le nom du modèle résultant est choisi. Nous proposons de l'appeler *Modèle logique*. Ultérieurement, si le modèle conceptuel est modifié, le nouveau modèle logique devrait être généré en choisissant l'option de mettre à jour le dernier modèle généré, sans mention du SGBD.

La figure 5-23 montre le résultat de la génération. Le panneau de gauche fait maintenant référence aux deux modèles qui vont cohabiter dans le même espace de travail. Il suffit de cliquer sur le nom du modèle pour l'afficher dans le panneau de droite.

FIGURE 5-23 **Modèle logique généré**

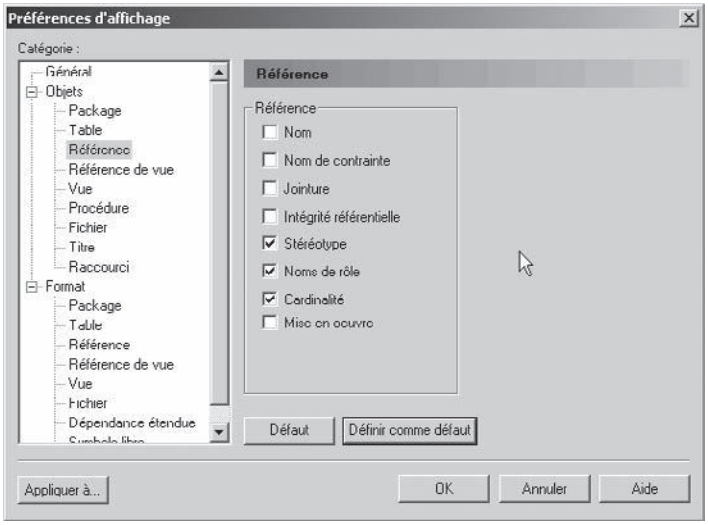


Il y a lieu de changer les paramètres d'affichage du modèle logique. Par défaut, les liens entre les tables affichent un nom de référence codé. Cet affichage alourdit le modèle. Il est plutôt souhaitable de voir les multiplicités des liens. Pour procéder au changement à l'affichage, on doit faire appel à la commande :

`Outils->Préférences d'affichage…`

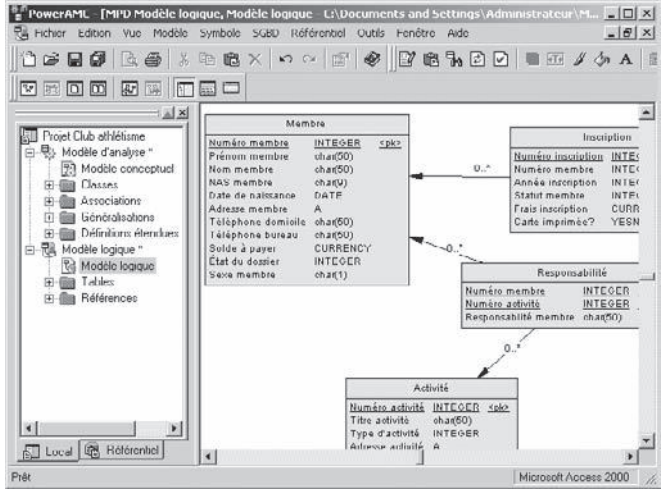
Choisir l'objet **Référence** dans le panneau de gauche, puis cocher l'option **Cardinalité** et retirer l'option **Nom de contrainte** comme le montre la figure 5-24. Cliquer ensuite le bouton *Définir comme défaut*.

FIGURE 5-24 **Préférences d'affichage du modèle logique**



Dans le contexte d'un modèle logique, le mot cardinalité est synonyme de multiplicité. En cochant cette option, les liens entre les tables seront affichés avec leurs multiplicités. La flèche qui est utilisée pour représenter un lien, pointe toujours vers la table mère.

FIGURE 5-25 **Effet du changement des préférences d'affichage du modèle logique**



Le lecteur constatera que PowerAMC n'affiche que les multiplicités du côté de la table fille comme le montre la figure 5-25. Ce choix des concepteurs de l'outil peut se justifier car, dans un modèle logique, les multiplicités du côté de la table mère sont soit 1..1 soit 0..1.

Dans le cas 1..1, PowerAMC INSCRIT la propriété **Obligatoire**, soit une contrainte *Non nul*, à la colonne clé étrangère assurant le lien. Comme nous l'avons vu au chapitre 3, c'est la convention pour mettre en œuvre une multiplicité minimale à 1 dans le modèle physique. PowerAMC met en œuvre cette contrainte dès le niveau logique.

Dans le cas 0..1, PowerAMC N'INSCRIT PAS la propriété **Obligatoire** dans la colonne clé étrangère assurant le lien. C'est la convention pour mettre en œuvre une multiplicité minimale de 0 dans le modèle physique.

Nous illustrons les possibilités offertes par PowerAMC pour générer un modèle logique grâce à deux séries de modèles conceptuels. D'abord, avec des MCD dont les modèles logiques dérivés sont à peu près conformes aux règles de dérivation abordées au chapitre 2 et n'exigeant que des ajustements mineurs. Puis ensuite, à l'aide de MCD montrant les limites de l'outil car le modèle logique dérivé doit subir des ajustements majeurs.

Ajustements mineurs au modèle logique

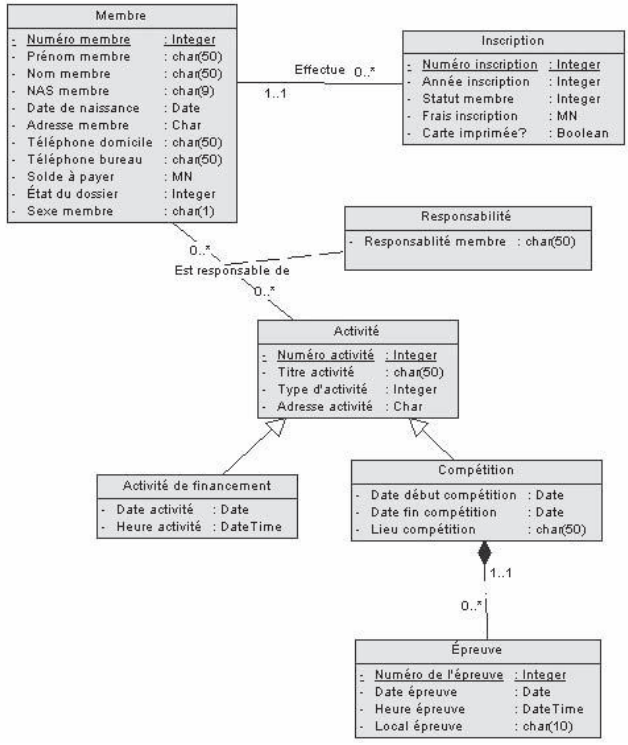
Le modèle conceptuel 5-26 est un bon exemple permettant de démontrer la validité du modèle logique généré par PowerAMC. Il comporte des associations parmi les types les plus répandus dans un modèle conceptuel : binaire un à plusieurs, binaire plusieurs à plusieurs, avec ou sans entités d'association, associations de composition et d'héritage.

Le modèle dérivé du modèle conceptuel est conforme à la théorie et reflète une application judicieuse des règles de dérivation. Les tables dérivées, le choix des clés primaires **<PK>** et la présence des clés étrangères **<fk>** qui assurent les liens entre les tables, respectent en tout point la théorie.

La figure 5-27 montre le modèle logique généré à partir de 5-26. Peu de changements seront requis pour le rendre conforme à la théorie.

Le premier ajustement porte sur l'application de l'intégrité référentielle en ajout, en mise à jour et en suppression. La théorie veut qu'une table dérivée d'une association plusieurs à plusieurs comporte une clé primaire composée des clés primaires des tables liées. Ces clés étrangères doivent porter une contrainte illustrée par la mention {Cascade} assurant l'intégrité référentielle en ajout, en mise à jour et en suppression. Or PowerAMCproduit une mention d'intégrité référentielle en ajout seulement. Le modélisateur devra

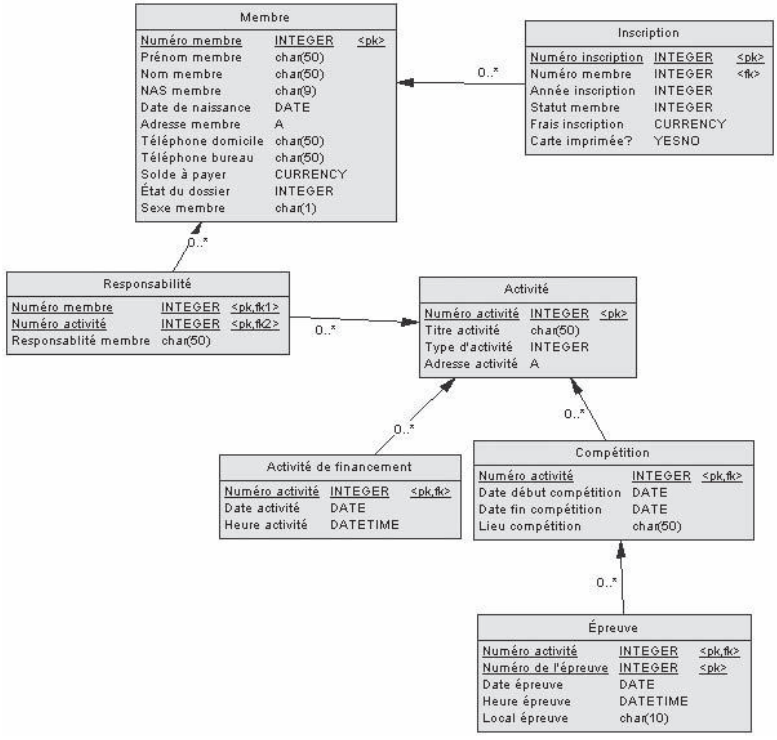
FIGURE 5-26 **Modèle conceptuel produit avec PowerAMC**



revoir dans le modèle logique tous les liens découlant d'une association plusieurs à plusieurs, d'une association d'héritage ou de composition. Pour ce faire, il suffit de faire un double-clic sur le lien (appelé une *référence* dans PowerAMC), de choisir l'onglet *Intégrité* et de cocher les options **Cascade** sous *Contrainte de modification* et **Cascade** sous *Contrainte de suppression* comme le montre la figure 5-28. Le modélisateur a fait un double clic sur le lien entre la table **Responsabilité** et **Membre**. Ce lien est dérivé de l'association **Est responsable de** du type *plusieurs à plusieurs*. L'option **Cascade** représente une contrainte d'intégrité référentielle en ajout, mise à jour et suppression sur le lien entre la table **Responsabilité** et **Membre**. Le modélisateur devra faire en plus le choix de cette option sur le deuxième lien dérivé de **Est responsable de**, soit le lien entre **Responsabilité** et **Activité**.

Le lien entre **Responsabilité** et **Membre** et celui liant **Responsabilité** et **Activité** auront alors leurs propriétés d'intégrité ajustées pour assurer la mise à jour et la suppression en cascade. Il devra en être de même pour le lien

FIGURE 5-27 **Modèle logique généré avec PowerAMC pour le modèle conceptuel 5-26**



Activité de financement et **Activité**, pour le lien entre **Activité** et **Compétition**, deux liaisons découlant des associations d'héritage, et pour le lien entre **Compétition** et **Épreuve** découlant d'une composition.

Le deuxième ajustement concerne les multiplicités 0..***** dérivées des associations d'héritage qui devraient être 0..1 selon la théorie. Ce changement s'opère à travers l'onglet *Intégrité* par la zone de liste déroulante **Cardinalité**.

Si le modélisateur souhaite appliquer à une clé primaire un type de données à génération automatique de valeurs, il a le loisir de le faire au niveau logique en modifiant le type de données de la colonne représentant la clé primaire de la table. Pour ce faire, on doit effectuer un double clic sur la table. Une fenêtre affiche alors les propriétés de la table. Sous l'onglet *Colonnes*, il est possible de remplacer le type de données de la clé primaire. C'est le cas de **Numéro membre** où le type COUNTER est choisi à travers une zone de liste déroulante, tel qu'illustré à la figure 5-29.

FIGURE 5-28 **Ajustement des contraintes d'intégrité sur le lien Responsabilité->Membre**

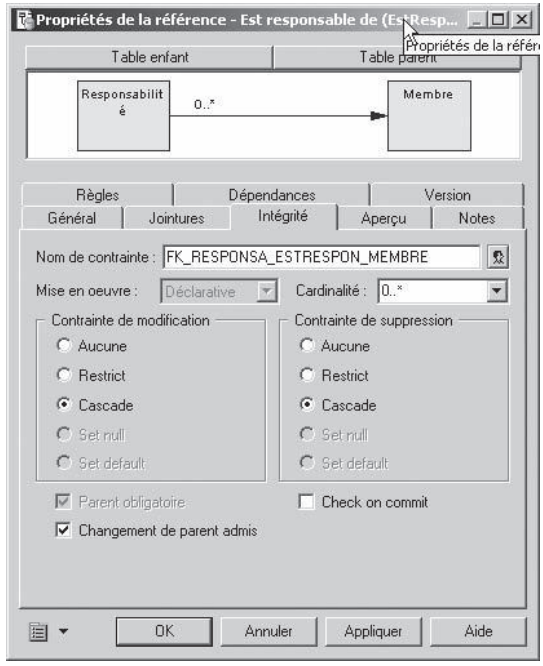
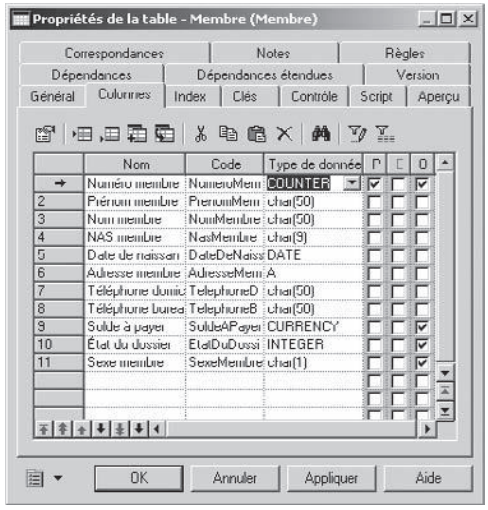


FIGURE 5-29 **Changement du type de données de la clé primaire de la table Membre pour COUNTER (génération automatique de valeurs)**



Le modélisateur peut consulter sous l'onglet *Colonnes* d'autres propriétés de la colonne. **P** indique qu'il s'agit d'une clé primaire. **O** indique la présence obligatoire de valeurs. Ces options découlent du modèle conceptuel et le modélisateur ne doit pas les modifier. L'option **O** est cochée pour les colonnes **Solde à payer**, **État du dossier** et **Sexe membre** car, au niveau conceptuel, les attributs correspondants possèdent une contrainte *{Non nul}*.

Ajustements majeurs au modèle logique

Dans le cas des associations *un à un* présentes au MCD, PowerAMC génère un modèle logique où certains liens entre les tables sont en complète contradiction avec la théorie. En effet, avec PowerAMC, toute association *un à un* entre une entité A et une entité B génère au niveau logique un lien de la table A vers B **et** un lien de B vers A. Ce qui pose des problèmes de références circulaires, entraîne l'impossibilité d'appliquer l'intégrité référentielle et de générer le modèle physique. L'intégrité référentielle exige, si on ajoute un enregistrement dans la table A, qu'un enregistrement correspondant soit présent dans B. S'il existe un lien de B vers A avec intégrité référentielle, cela implique, pour ajouter un enregistrement dans B, qu'il faut que l'enregistrement existe dans A. Ce qui rend impossible l'ajout d'enregistrements à la fois dans A et dans B lorsque l'intégrité référentielle est requise.

Pour éviter cette situation, le modélisateur doit effacer un des deux liens générés par PowerAMC entre les tables A et B.

Considérons le MCD de la figure 5-30 et le modèle logique généré à l'aide de PowerAMC présenté à la figure 5-31.

FIGURE 5-30 **Association *un à un* entre les entités Client et Carte de crédit**

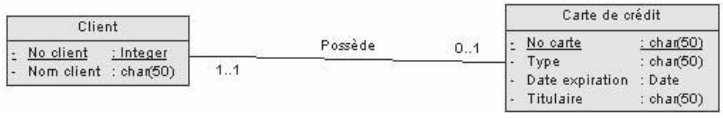
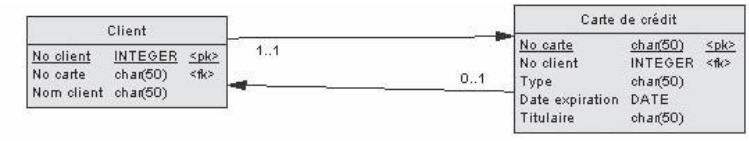


FIGURE 5-31 **Tables et liens générés par PowerAMC pour le MCD 5-29**

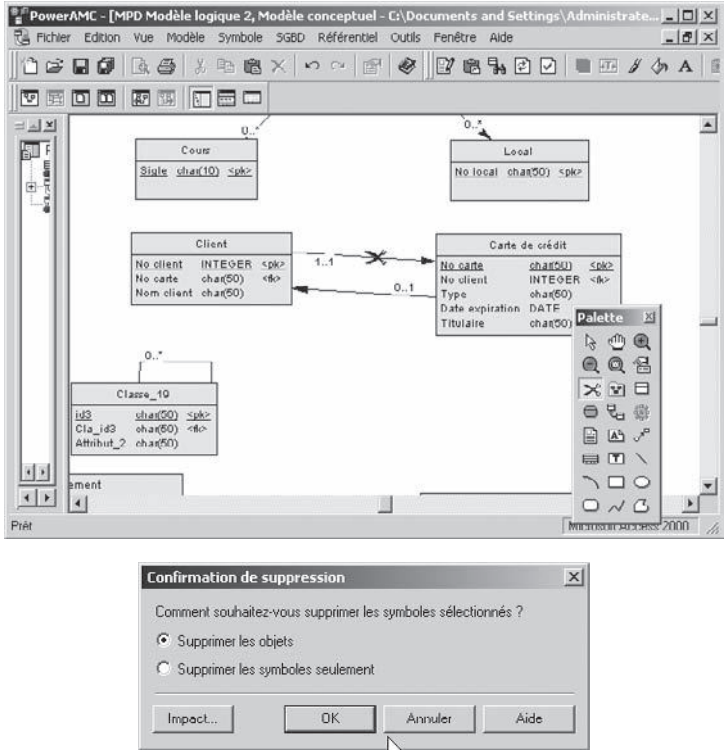


Il est impératif de retirer le lien redondant. La théorie veut que toute association *un à un* produise une seule table fille qui sera dérivée de l'entité du côté de la multiplicité 0..1. Cela implique que le lien de la table **Client** vers la table **Carte de crédit**, présent dans le modèle logique à la figure 5-30 et dont la multiplicité est 1..1, doit disparaître.

Pour y arriver, il faut procéder en trois temps :

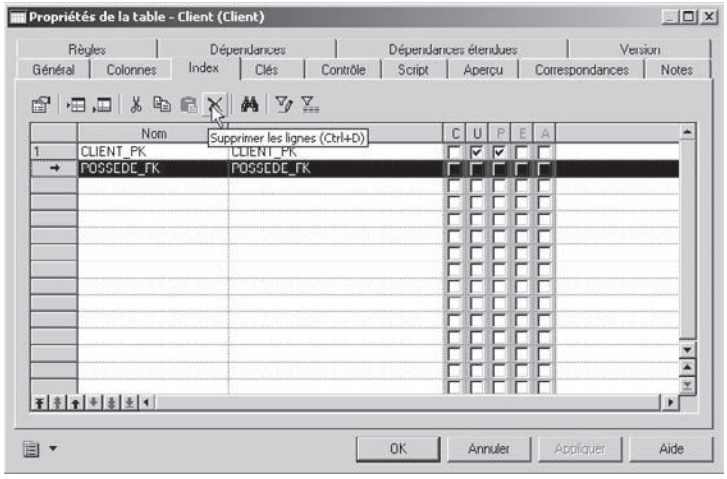
1. Supprimer le lien avec l'outil **Ciseaux** de la palette et confirmer la suppression (figure 5-32) ;
2. Double clic sur la table fille du lien effacé, ici **Client**, puis sous l'onglet *Index*, supprimer l'index que le lien avait généré (figure 5-33) ;
3. Double clic sur la table mère du lien effacé, ici **Carte de crédit**, puis sous l'onglet *Index*, modifier l'index que le lien avait généré en cochant l'option **U** (*unique,* soit clé étrangère sans doublon) (figure 5-35).

FIGURE 5-32 **Suppression du lien Client->Carte de crédit**



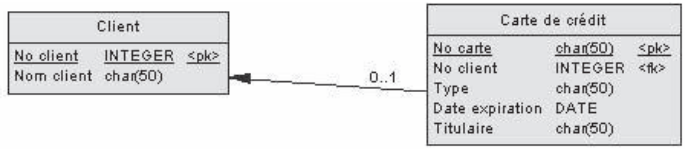
L'index qui doit être effacé dans la table fille porte le nom de l'association donné au MCD suivi de _**FK**. Pour la table **Client**, il s'agit de l'index nommé **Possede_FK**.

FIGURE 5-33 **Suppression de l'index Possede_FK dans la table Client**



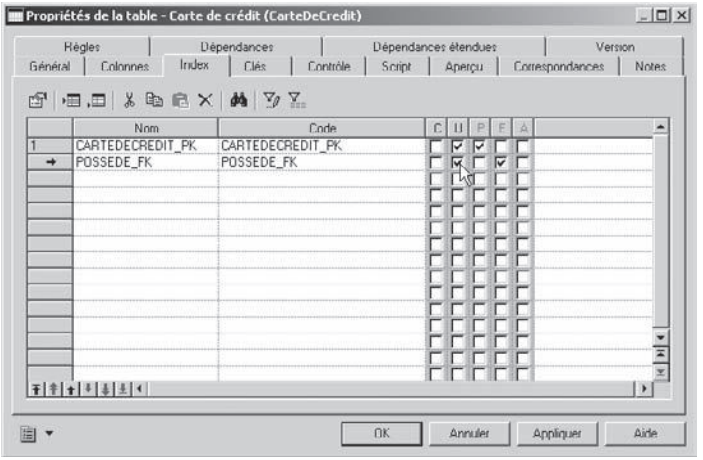
La figure 5-34 montre l'état du MCD à la suite de la suppression du lien et de l'index.

FIGURE 5-34 **Modèle logique résultant des suppressions dans le MCD 5-30**



Le processus se termine avec la modification d'un index dans la table **Carte de crédit**, celui qui porte le même nom que celui supprimé dans la table **Client**. Cet index doit avoir la propriété *Unique* (**U**) (figure 5-34). La propriété U réalise la multiplicité maximale 1 du côté de la table **Carte de crédit**.

FIGURE 5-35 **Modification de l'index Possede_FK dans la table mère Carte de crédit**



Génération du modèle physique et création de la BD

Un modèle physique est généré lorsqu'un modèle logique est actif à l'écran, auquel cas la commande suivante peut être effectuée :

`SGBD->Générer la base de données…`

Deux possibilités s'offrent au modélisateur : génération d'un script qui devra être lu par le SGBD ou création immédiate de la BD. Il est préférable de procéder par génération de script dans un fichier. Ce script pourra être consulté et modifié au besoin.

PowerAMC exige de réaliser deux scripts qui seront produits dans la séquence suivante :

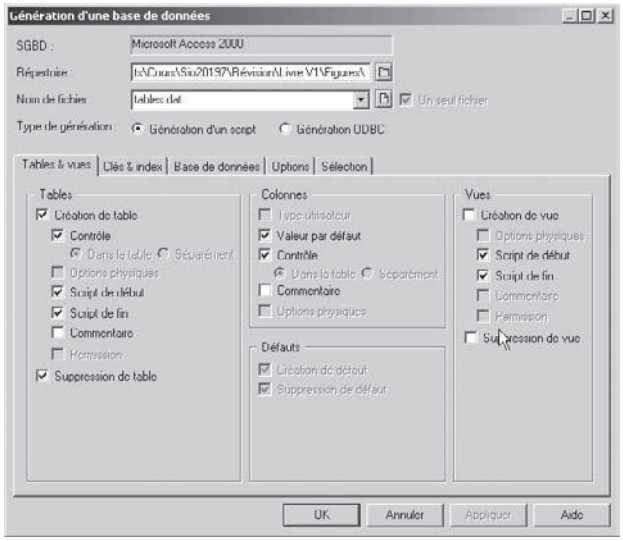
1. Création des tables et des index pour les clés primaires et étrangères ;
2. Création des contraintes d'intégrité référentielle sur les clés étrangères.

Les scripts seront générés dans deux fichiers différents de type *.dat*.

Création des tables et des index

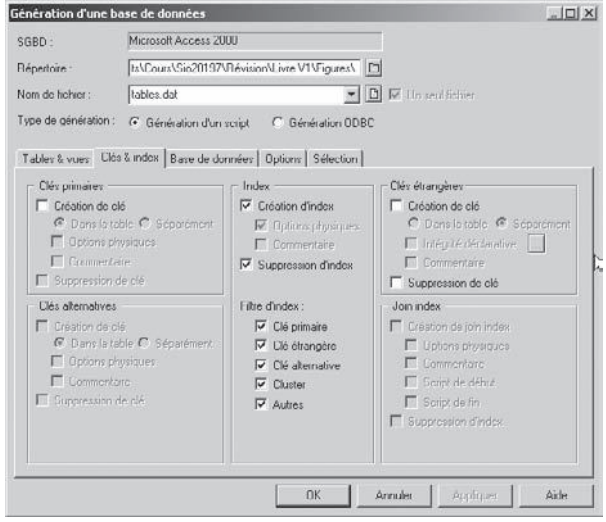
Le premier fichier script généré pourrait se nommer *tables.dat*. Il est produit en complétant le champ **Nom de fichier** et en sélectionnant le répertoire où sera créé le script. Par ailleurs, on doit choisir des options sous l'onglet *Tables & vues* selon les exigences de la figure 5-36 et des options de l'onglet *Clés & index* en conformité avec la figure 5-37.

FIGURE 5-36 **Options de la première génération sous l'onglet Tables & vues**



Il est très important que les options **Valeur par défaut** et **Contrôle** soient cochées pour que le script réalise la valeur par défaut de la colonne et les contraintes d'intégrité sémantique qui ont été données à l'attribut correspondant dans le MCD.

FIGURE 5-37 **Options de la première génération sous l'onglet Clés & index**



L'option filtre d'index **Clé étrangère** va générer le script nécessaire pour réaliser une colonne qui est clé étrangère et dont les valeurs doivent être uniques (Contrainte *{Unique}*). Si cette option est choisie, et c'est important qu'elle le soit si un modèle logique comporte des liens *un à un* entre des tables, PowerAMC ne permet pas de choisir l'option **Création de clé**. Les options **Clé étrangère** et **Création de clé** sont considérées mutuellement exclusives par PowerAMC. Or c'est précisément **Création de clé** qui réalise les contraintes d'intégrité référentielle du modèle logique. Voilà pourquoi il faut générer un deuxième script qui assurera par ailleurs la réalisation des contraintes d'intégrité référentielle.

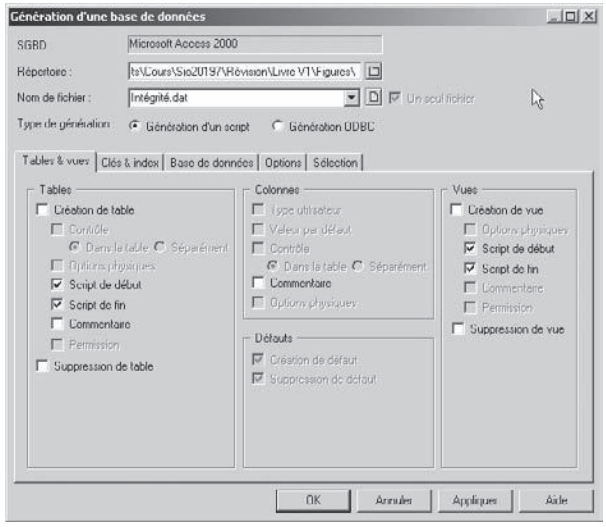
Création des contraintes d'intégrité référentielle

La deuxième génération de script fait à nouveau appel à la commande :

`SGBD->Générer la base de données…`

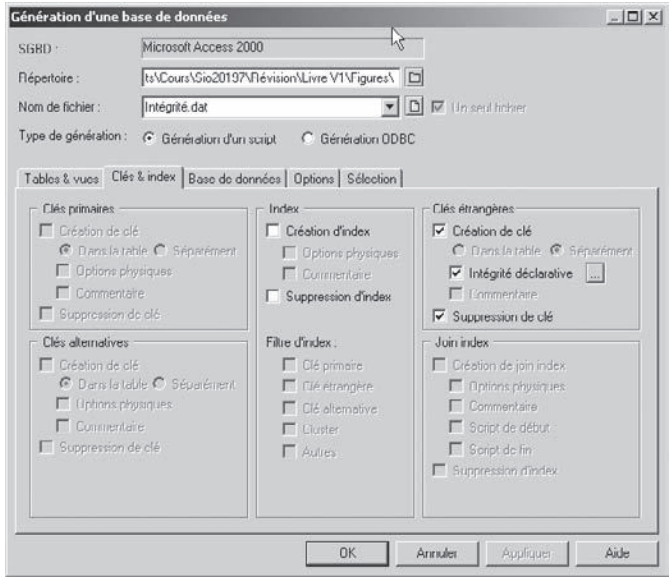
Le deuxième fichier script généré pourrait se nommer *Intégrité.dat*. Il est produit en complétant le champ **Nom de fichier** et en sélectionnant le répertoire où sera créé le script. On doit par ailleurs choisir des options sous l'onglet *Tables & vues* selon les exigences de la figure 5-38 et des options de l'onglet *Clés & index* en conformité avec la figure 5-39.

FIGURE 5-38 **Options de la deuxième génération sous l'onglet Tables & vues**



L'option **Création de table** ne doit pas être cochée car les tables auront été créées par le premier script.

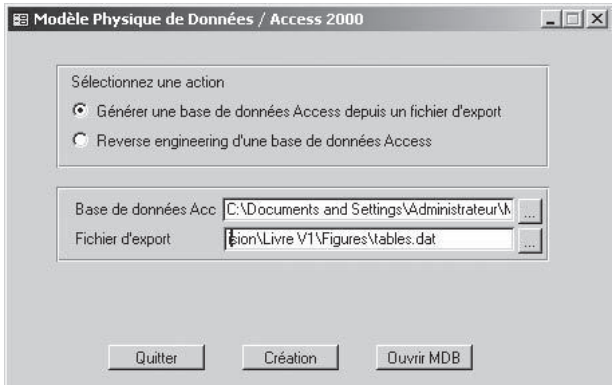
FIGURE 5-39 **Options de la deuxième génération sous l'onglet Clés & index**



L'option **Création d'index** ne doit pas être cochée car les index auront été créés par le premier script.

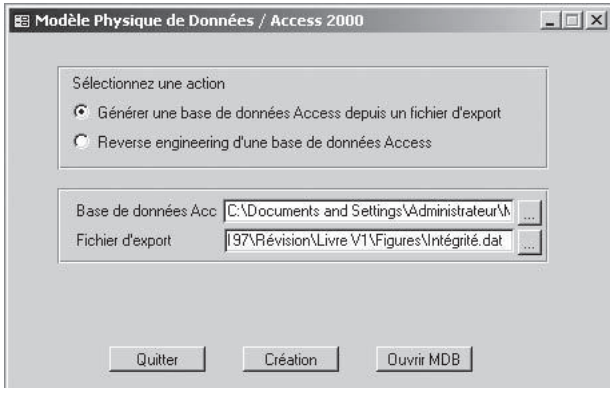
L'exécution des scripts pour le SGBD Access doit se faire en ouvrant une BD appelée **Access2K** disponible dans le répertoire **Outils** de l'application Sybase PowerAMC. L'ouverture de la BD provoque l'affichage de la fenêtre donnée à la figure 5-40.

FIGURE 5-40 **Exécution du premier script**



Le champ **Base de données Access** donne le chemin d'accès et le nom de la BD dans laquelle le modèle physique sera généré. Si la BD n'existe pas déjà, une BD vide sera d'abord créée sous le nom fourni. Le champ **Fichier d'export** précise le chemin d'accès et le nom du script à exécuter (Tables.dat). L'exécution du script est lancée en cliquant sur le bouton *Création*. Le modélisateur peut maintenant procéder à l'exécution du deuxième script.

FIGURE 5-41 **Exécution du deuxième script**



Le champ **Base de données Access** doit demeurer réglé avec la même BD qu'à l'étape précédente. Cependant le champ **Fichier d'export** doit donner le chemin d'accès et le nom du deuxième script à exécuter (Intégrité.dat). L'exécution du script est lancée en cliquant sur le bouton *Création* après quoi, en cliquant sur *Ouvrir MDB*, on peut avoir accès à la BD générée à l'aide des deux scripts.

Le bilan

La principale lacune de PowerAMC réside dans son incapacité de traiter adéquatement les associations *un à un* présentes au modèle conceptuel et à partir duquel un modèle logique est généré. Or de telles associations sont peu nombreuses dans un MCD. De plus, comme nous l'avons vu au chapitre 2, les liens *un à un* au modèle logique peuvent être éliminés en fusionnant les tables si les multiplicités sont du type 1..1-0..1 ou 1..1-1..1. Le modélisateur pourrait effectuer ces fusions dans le modèle logique avant de générer le modèle physique.

Si le modèle logique ne comporte pas de liens *un à un*, un seul script est nécessaire. Dans un tel cas, il n'est pas nécessaire en effet de cocher l'option **Filtre d'index Clé étrangère**. Cette option n'est requise qu'en présence de liens *un à un*. Puisque l'option **Filtre d'index Clé étrangère** n'a pas à être cochée, PowerAMC accepte de générer les contraintes d'intégrité référentielle dans le même script (Options **Création de clé**, **Intégrité déclarative**).

Même si un modèle conceptuel ne possède pas d'associations *un à un*, des ajustements restent nécessaires. Les principaux ajustements concernent l'application des contraintes d'intégrité référentielle en mise à jour et en suppression.

PowerAMC n'en demeure pas moins un outil très puissant car il permet au modélisateur de sauver un temps précieux en automatisant les tâches de dérivation des modèles. Il offre au modélisateur des outils de rétroconception dont nous n'avons pas fait état jusqu'ici. Par rétroconception (en anglais *reverse engineering*) nous entendons la possibilité de générer un modèle logique à partir d'une BD réelle, ou un modèle conceptuel à partir d'un modèle logique. Notre expérience à cet égard n'est pas très positive. Ces outils fonctionnent bien sur de petits modèles. Les modèles de grande taille (plusieurs tables et plusieurs liens) produisent des résultats approximatifs qui demandent des ajustements.

Notons en terminant que PowerAMC permet de créer un MOO de type *Diagramme de cas d'utilisation*. Nous reportons le lecteur à la figure 5-2 où est illustrée la fenêtre permettant de créer un nouveau MOO. La création d'un diagramme de cas d'utilisation s'effectue en sélectionnant ce type de diagrammes UML à travers la liste de choix **Premier diagramme**. La réalisation d'un diagramme de cas d'utilisation avec PowerAMC est d'une grande simplicité.

WIN'DESIGN

Tout comme PowerAMC, Win'Design permet la réalisation de modèles conceptuels de données à l'aide de la notation Merise. Pour ce faire, le modélisateur doit faire appel à un des quatre modules du logiciel, soit le module **Database**.

S'il souhaite produire un MCD en notation UML, il doit cependant faire appel au module **Object**. C'est ce volet du logiciel que nous avons évalué. Le module **Object** permet par ailleurs de générer un MCD en notation Merise à partir de la version UML du modèle conceptuel ou, bien sûr, un modèle logique de données à partir du MCD réalisé en notation UML.

Win'Design offre au modélisateur la possibilité de dériver un modèle physique à partir d'un modèle logique pour une large panoplie de SGBD, dont MS Access. Tout comme dans le cadre de l'évaluation de PowerAMC, nous allons traiter à tour de rôle de la création d'un MCD en notation UML et des modalités de dérivation des modèles logique et physique à partir du MCD.

Nous avons évalué la version 7.5 de Win'Design.

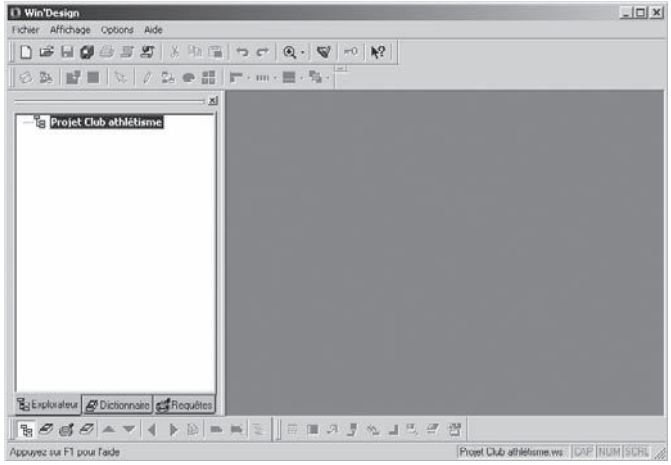
Réalisation d'un modèle conceptuel de données avec Win'Design

Tous les modèles réalisés dans le cadre du même projet sont conservés dans un *Espace de travail*. La première fois que le logiciel est lancé, un espace de travail vide est présenté portant le nom *Espace de travail*. Ce nom peut être changé en cliquant sur l'étiquette du nom. La figure 5-42 montre un espace de travail vide portant le nom *Projet Club athlétisme*.

Il est fortement suggéré de sauvegarder l'espace de travail initial, bien que vide, de manière à créer dans un répertoire approprié le fichier de type *.ws* qui regroupe les références à tous les modèles du projet :

`Fichier->Espace de travail->Enregistrer sous…`

FIGURE 5-42 **Fenêtre de démarrage de Win'Design**



Tout nouveau modèle doit être créé dans l'espace de travail par la commande :

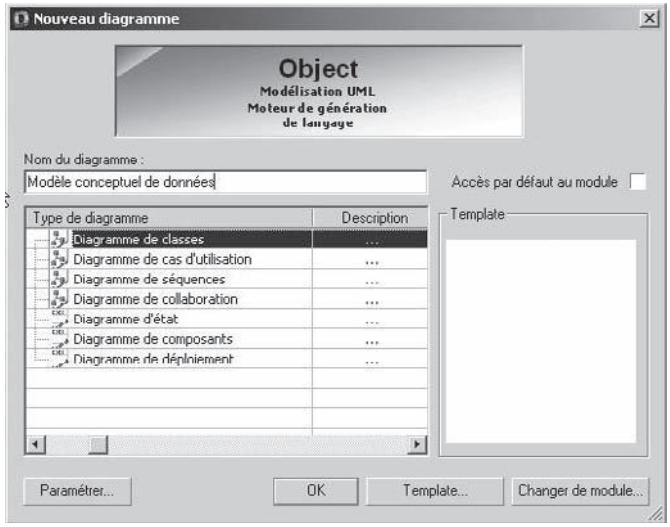
`Fichier->Nouveau…`

La figure 5-43 montre les quatre modules qui s'offrent alors à l'utilisateur. Le modélisateur doit choisir le module **Object** pour réaliser un MCD avec la notation UML.

FIGURE 5-43 **Création d'un nouveau modèle dans l'espace de travail**

Le module **Object** sera utilisé pour la réalisation de plusieurs types de diagrammes UML. Un MCD réalisé avec la notation UML se fait par le biais d'un *diagramme de classes* tel que l'illustre la figure 5-44. Il y lieu de donner au modèle un nom approprié dès cet instant (zone **Nom du diagramme**).

FIGURE 5-44 **Création d'un MCD en UML par le biais d'un diagramme de classes**

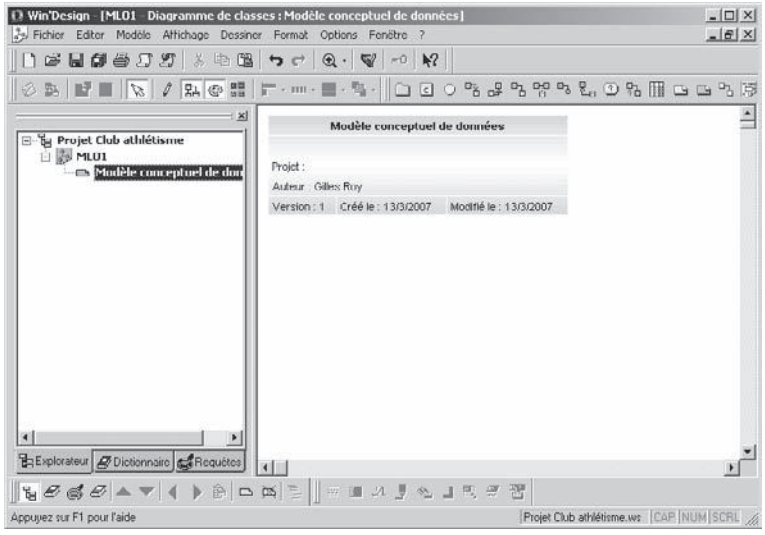


Le modélisateur devra choisir par ailleurs le langage cible par le biais d'une fenêtre affichée lorsque le bouton *OK* est pressé. Il est conseillé de choisir **Analyse**, car un MCD se situe au niveau de l'analyse des besoins sans égard au choix du langage de programmation.

La réalisation d'un modèle conceptuel en UML doit impérativement se faire avec un modèle de type MLO (Modèle Langage Objet). Le *Diagramme de classes* appartient à cette catégorie de modèles tout comme le *Diagramme de cas d'utilisation* qui sera ajouté à l'espace de travail ultérieurement. La figure 5-45 montre l'état initial de l'espace de travail avec à droite un panneau où seront créées les entités et les associations du MCD.

Pour ce faire, une barre d'outils, située juste au-dessus du panneau de droite, est disponible. Le tableau 5-4 reprend certains éléments de la barre d'outils qui sont pertinents à la réalisation d'un modèle conceptuel de données.

FIGURE 5-45 **Création d'un MCD en UML à l'aide d'un diagramme de type MLO**



Le tableau 5-4 montre les outils pertinents à la réalisation d'un MCD. On constate qu'une association de composition ne fait pas appel à un outil distinct pour sa création comme c'est le cas pour l'association d'héritage. En effet, une composition doit être créée avec l'outil *Association*. Le choix du type d'association *composition* sera finalisé au moment où seront établies les multiplicités de l'association du côté de l'entité composite, comme nous le verrons plus loin.

TABLEAU 5-4 **Modèle conceptuel et outils pertinents**

Icône	Terminologie UML	Dénomination au niveau conceptuel
	Classe	Entité
	Association	Association binaire ou de degré supérieur
	Généralisation	Association d'héritage
	Composition	Association de composition
	Contrainte	Contrainte inter-association
	Lien formel	
	Classe association	Entité d'association

Win'Design Version 7.5 pour Windows impose les limites suivantes quant à la réalisation d'un modèle conceptuel de données en UML :

- des contraintes inter-association peuvent être exprimées en UML avec les outils **Contrainte**  et **Lien formel**  ; comme ces contraintes sont exprimées librement et sans cadre formel, le modèle logique de données dérivé n'en tient nullement compte ; il devra être modifié par le modélisateur pour qu'elles soient implantées le cas échéant.
- les contraintes d'intégrité sémantique que l'on peut formuler dans le modèle conceptuel sont les suivantes : liste de valeurs acceptables, valeur minimum, valeur maximum, valeur par défaut ; les autres contraintes sont formulées textuellement et ne sont pas prises en charge lors du passage au modèle logique.

Avant de procéder à la réalisation d'un premier modèle conceptuel de données, il importe de fixer des paramètres qui seront appliqués par défaut à tout modèle faisant appel au diagramme de classes. Ces paramètres concernent essentiellement les préférences d'affichage du modèle. La plupart des préférences d'affichage des diagrammes de classes ne concernent pas un MCD. Par exemple, il n'y a pas lieu d'afficher des icônes représentant la visibilité des attributs d'une entité dans MCD, la visibilité n'est pertinente que pour un attribut dans une classe.

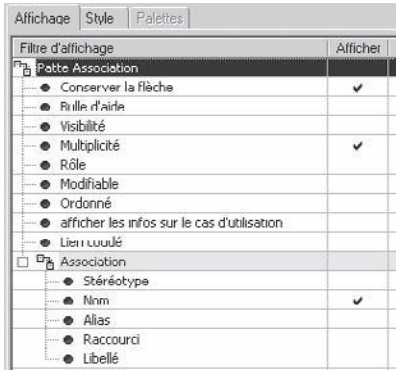
`Options->Profil standard->Modifier…`

Cette commande permet de régler l'affichage des classes et des associations.

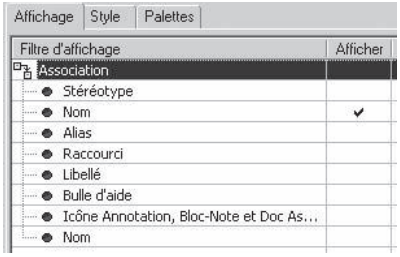
Déployer **MLO-Modèle langage objet** et sélectionner **Diagramme de classes**. Presser le bouton *Options graphiques…*

Nous proposons de procéder à certains réglages dans l'ordre suivant :

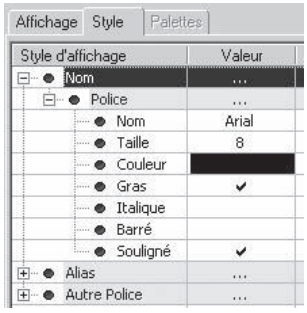
1. N'afficher que le nom de l'association et ses multiplicités. Dans le panneau de gauche, déployer **Liens** et sélectionner **Patte association**, puis régler ainsi les options du panneau de droite sous l'onglet *Affichage* :



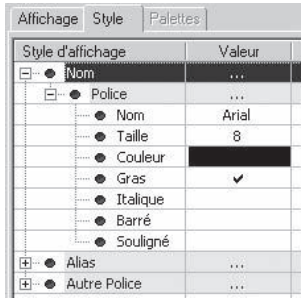
2. Dans le panneau de gauche, sélectionner **Association** et régler ainsi les options du panneau de droite sous l'onglet *Affichage* :



3. L'identifiant apparaîtra en caractères gras et sera souligné. Dans le panneau de gauche, sélectionner **`Attribut->identifiant`** et régler ainsi les options du panneau de droite sous l'onglet *Style* :

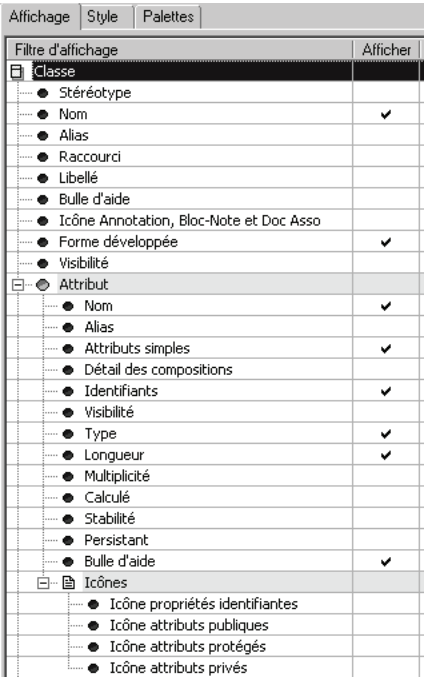


4. Tout attribut obligatoire est en gras. Dans le panneau de gauche, sélectionner **`Attribut->obligatoire`** et régler ainsi les options du panneau de droite sous l'onglet *Style*:



5. Pour chaque entité, l'identifiant sera affiché ainsi que le type et la longueur de chaque attribut. Dans le panneau de gauche, sélectionner **Classe** et régler ainsi les options du panneau de droite sous l'onglet *Affichage*:

6. Pour chaque entité d'association, l'identifiant sera affiché ainsi que le type et la longueur de chaque attribut. Dans le panneau de gauche, sélectionner **`Classe->Association`** et régler ainsi les options du panneau de droite sous l'onglet *Affichage*:



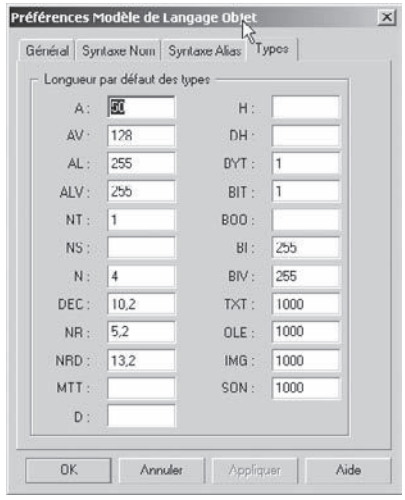
Tous ces paramètres seront enregistrés pour être conservés dans le fichier **Standard.cfg** géré par Win'Design. Désormais, ils n'auront plus à être réglés pour répondre aux exigences d'affichage d'un modèle conceptuel de données.

Par ailleurs, il y a lieu de fixer par défaut la longueur et la précision de la valeur d'un attribut selon le type de données qui lui sera affecté. Pour ce faire, exécuter:

`Options->Préférences->Modèle…`

Sous l'onglet *Types*, régler notamment le type A (caractère) à 50, le type NS (entier) à blanc, le type D (date) à blanc et le type MTT (monétaire) à blanc comme le montre la figure 5-46.

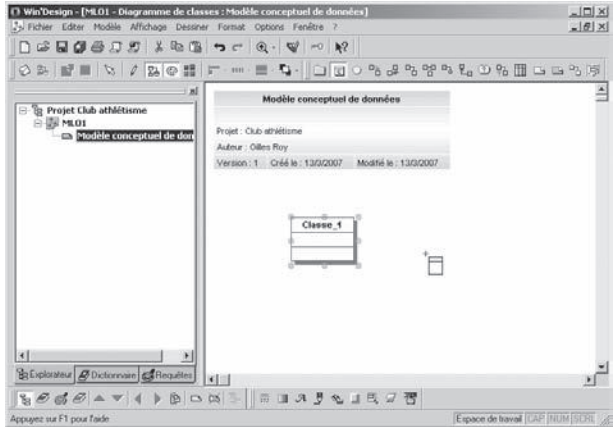
FIGURE 5-46 **Fixation des longueurs et précisions par défaut des attributs**



Création d'une entité

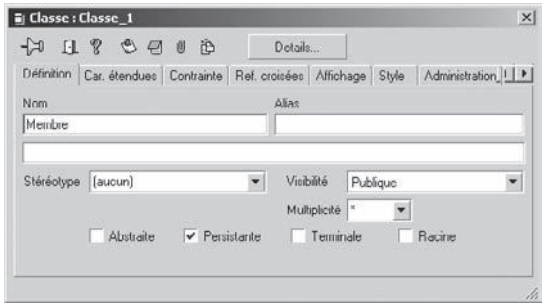
En vertu du tableau 5-4, une entité doit être créée avec l'outil classe . Il suffit de cliquer dans le panneau de droite pour y inscrire une ou des entités dont le nom donné par défaut débute par `Classe_` (Figure 5-47). Dès l'instant où une entité est créée, il suffit d'effectuer un double-clic sur celle-ci pour définir les propriétés de l'entité, notamment son nom, ses attributs, leur type de données, leurs contraintes d'intégrité et l'identifiant de l'entité.

FIGURE 5-47 **Création d'une entité**



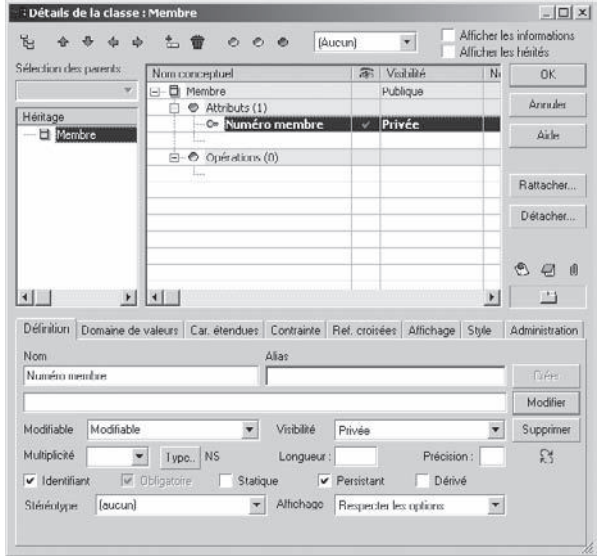
Comme l'illustre la figure 5-48, certaines propriétés de l'entité sont saisies dans un premier panorama. Le nom de l'entité doit être inscrit dans la zone **Nom**. L'option **Persistante** est cochée par défaut. Cette option va assurer la génération d'une table à partir de l'entité dans le modèle logique.

FIGURE 5-48 **Propriété de l'entité**



Dans un deuxième temps les attributs seront ajoutés. C'est par le biais du bouton *Détails…* que sont saisis les attributs de l'entité, leur type et leurs contraintes d'intégrité. Une deuxième fenêtre appelée *Détails de la classe* vient alors se superposer à la première, telle que décrite dans la figure 5-49.

FIGURE 5-49 **Saisie des attributs de l'entité Membre et de leurs types de données**



Les attributs sont ajoutés sous la rubrique *Attributs()*. L'onglet *Définition* comporte des zones pour ajouter ou modifier les propriétés d'un attribut. Un attribut qui agit comme identifiant devrait avoir l'option **Identifiant** cochée. Tout attribut doit être **Persistant** (Option définie par défaut). Si l'attribut est *Non nul*, **Obligatoire** doit être coché. Le choix du type de données de l'attribut se fait par le biais du bouton *Type…* . Un identifiant est automatiquement **Obligatoire** et sera souligné en vertu des préférences d'affichage données plus tôt.

Les types de données de base exposés au chapitre 1 sont repris dans le tableau 5-5 et mis en correspondance avec les types Win'Design offerts pour un modèle conceptuel. Win'Design offre plusieurs autres types de données prédéfinis dont le type *Monnaie*.

TABLEAU 5-5 **Types de données Win'Design dans un modèle conceptuel**

Type	Type Win'Design
`Integer`	Entier (NS)
`Real`	Décimal (DEC)
`String`	Caractère (A)
`Date`	Date (D)
`Boolean`	Booléen (BOO)
`enum{}`	Le type doit être choisi parmi les types précédents et la liste des valeurs possibles sera donnée par l'onglet ***Domaine de valeurs*** adjacent à l'onglet ***Définition***.

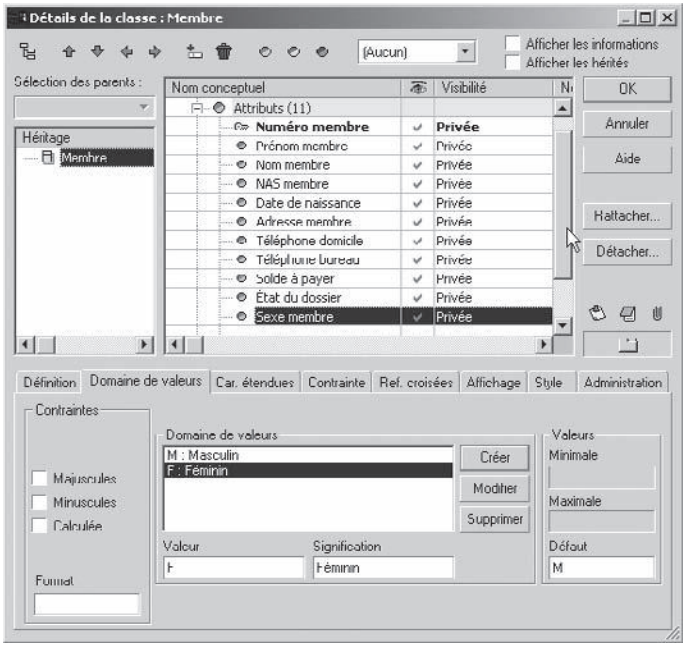
La figure 5-50 montre tous les attributs de l'entité **Membre**. On note que l'onglet *Domaine de valeurs* est activé de manière à faire voir les contraintes d'intégrité de l'attribut **Sexe membre** : Valeurs admissibles (M ou F) et Valeur par défaut (M).

Nous présentons dans le tableau 5-6 les contraintes d'intégrité sémantique que le modélisateur peut exprimer dans un modèle conceptuel réalisé avec Win'Design ainsi que l'onglet approprié.

TABLEAU 5-6 **Contraintes d'intégrité sémantique dans un modèle conceptuel**

Contrainte	Onglet	Zones concernées
Non nul	Définition	**Obligatoire** doit être coché
Unique	Définition	**Identifiant** doit être coché
Valeur par défaut	Domaine de valeurs	**Défaut** doit être complété avec la valeur par défaut
Valeur minimum	Domaine de valeurs	**Minimale**
Valeur maximum	Domaine de valeurs	**Maximale**
Valeurs admissibles	Domaine de valeurs	**Domaine de valeurs**, zone **Valeur,** une valeur admissible, zone **Signification**, l'étiquette associée à la valeur si celle-ci est codée

FIGURE 5-50 **Attributs de l'entité Membre**



Création d'une association binaire

La barre d'outils comporte l'icône *Association* qui, une fois sélectionnée, permet de créer une association entre deux entités en cliquant sur l'une d'elle, en glissant le curseur vers la deuxième tout en relâchant le bouton, puis en cliquant sur cette dernière.

La figure 5-51 montre le résultat de la création d'une association binaire entre les entités **Membre** et **Activité**. Par défaut l'association porte un nom débutant par `Asso_` et elle possède des multiplicités 0..* et 0..* .

Il suffit de faire un double clic sur le nom de l'association pour changer son nom. On procède de la même manière pour changer les multiplicités. Un double clic sur une multiplicité ouvre une fenêtre permettant de choisir la multiplicité voulue sous l'onglet *Définition*.

Lorsqu'une association binaire porte une entité d'association, l'association doit d'abord être créée comme décrit ci-dessus. L'entité qui sera portée par l'association sera créée ensuite et en tout dernier lieu avec l'outil *Classe*

association . Le modélisateur doit cliquer sur l'entité que porte l'association, glisser le curseur vers cette association en relâchant la pression sur le bouton, puis cliquer sur le nom de l'association. Dès lors, une ligne pointillée se dessine entre l'entité et l'association. Il s'agit maintenant d'une entité d'association (Figure 5-52).

FIGURE 5-51 **Propriétés données par défaut à une association**

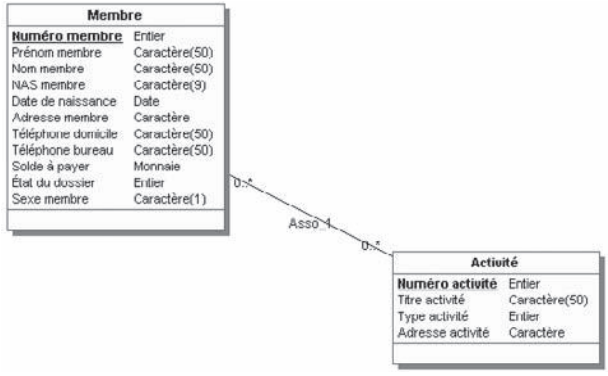
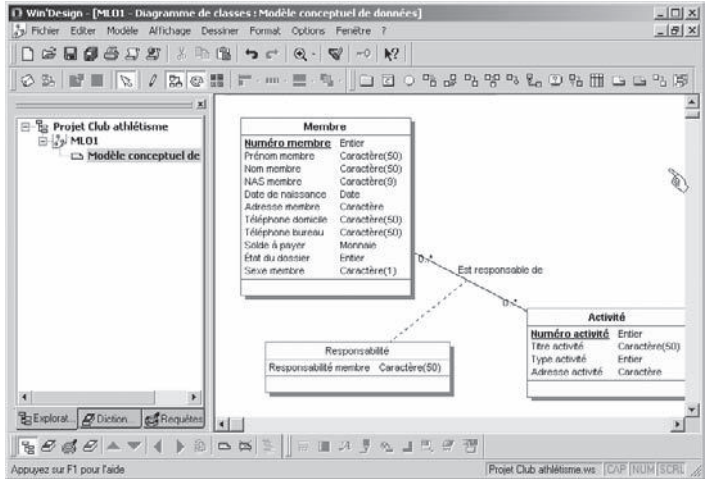


FIGURE 5-52 **Création d'une entité d'association Responsabilité**

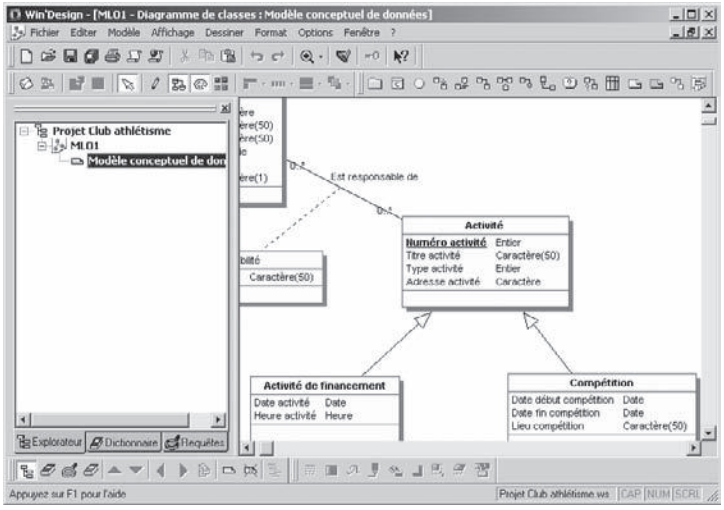


Création d'une association d'héritage ou de composition

L'association d'héritage se construit entre deux entités avec l'outil *Généralisation*. Il suffit de cliquer sur l'entité sous-type, de relâcher le bouton, de glisser le curseur vers l'entité supertype puis de cliquer sur celle-ci. Une association, portant le triangle pointé vers le supertype, est alors dessinée.

La figure 5-53 montre deux associations d'héritage créées sur la même entité supertype : **Activité**.

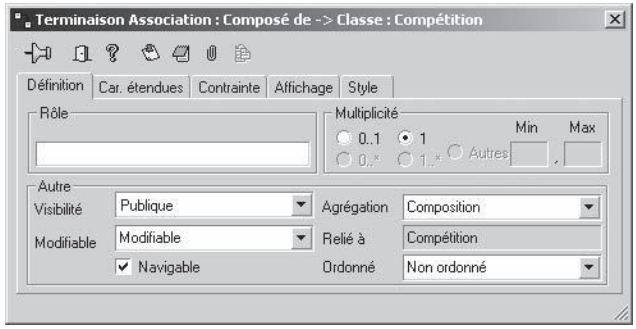
FIGURE 5-53 **Création de deux associations d'héritage**



Pour réaliser une association de composition, il suffit de créer entre l'entité composant et l'entité composite une association binaire comme nous l'avons décrit plus haut. Il est impératif de donner à l'association un nom qui indique explicitement qu'il s'agit d'une composition, par exemple : **Composé de**. De plus, il faudra changer les propriétés de la multiplicité du côté de l'entité composite pour y afficher le petit losange noir. Un double clic sur la multiplicité ouvre une fenêtre de propriétés qui doivent être réglées selon les exigences suivantes sous l'onglet *Définition* (Voir figure 5-54) :

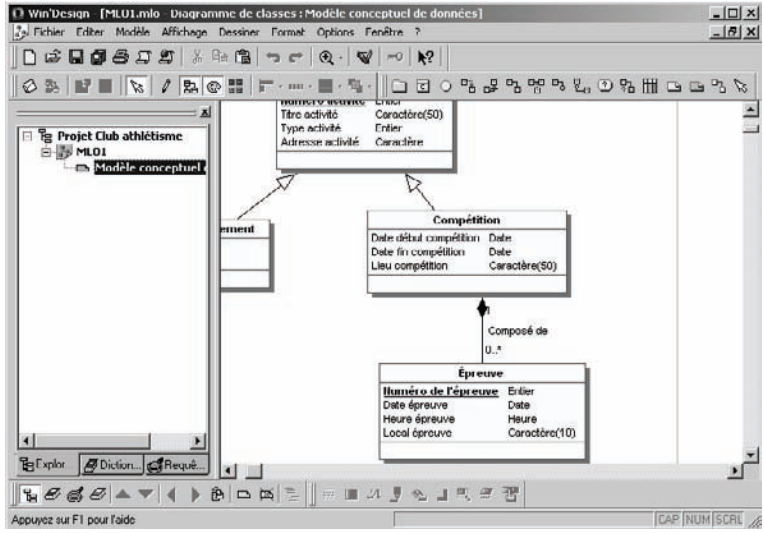
- Zone **Multiplicité** à 1
- Zone **Agrégation** réglée à *Composition* à travers la zone de liste

FIGURE 5-54 **Réglage des propriétés de multiplicité du côté du composite**



La figure 5-55 montre une association de composition appelée **Composé de** créée entre l'entité **Compétition** et l'entité composant **Épreuve**.

FIGURE 5-55 **Création d'une association de composition**



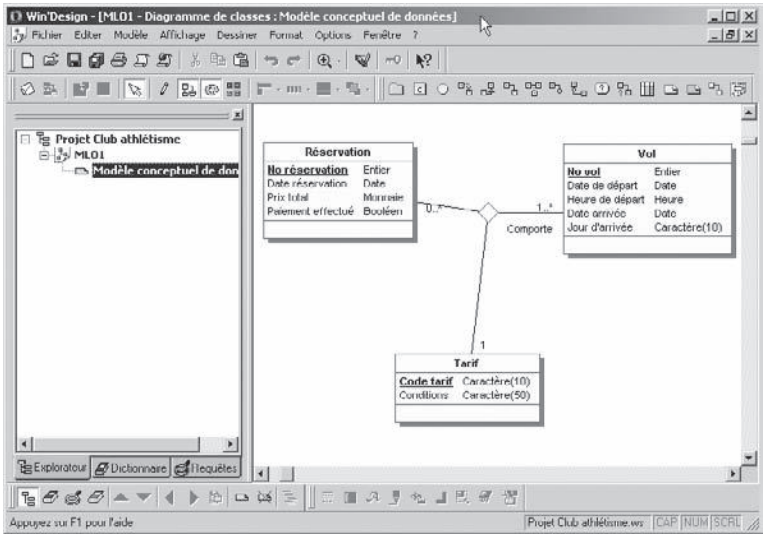
Création d'une association de degré supérieur

Win'Design permet au modélisateur de créer des associations de degré supérieur selon une démarche qui s'inspire de la création des associations binaires. Les entités associées sont créées dans un premier temps. Le modélisateur crée arbitrairement une association binaire avec deux des entités

grâce à l'outil *Association* comme décrit plus tôt. Une troisième entité sera associée en cliquant sur celle-ci, puis en glissant le curseur sur le nom de la première association, en relâchant le bouton et en en cliquant sur ce nom. Le losange représentant une association de degré supérieur est alors affiché sur l'association. Toute autre entité peut être ajoutée à l'association en cliquant sur celle-ci d'une part et sur le losange d'autre part. Finalement le nom de l'association et les multiplicités sont modifiés à tour de rôle en procédant à un double clic sur chacun d'eux.

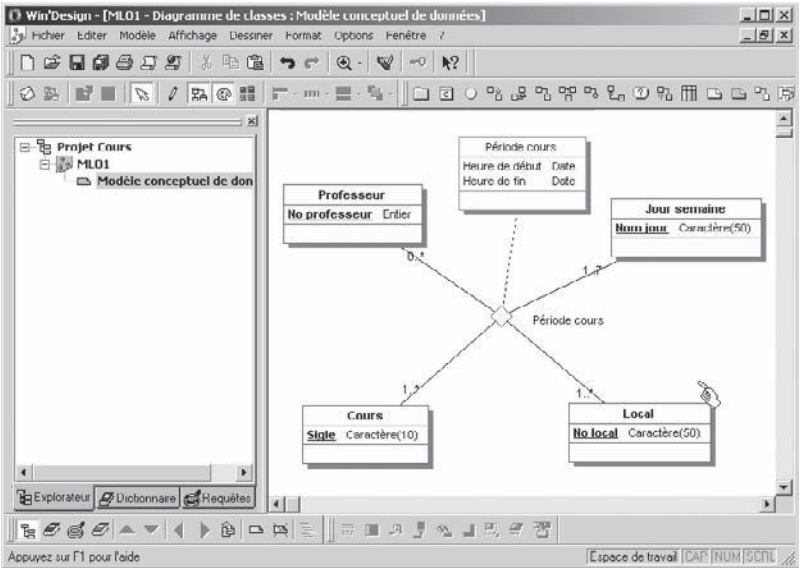
Le modèle conceptuel illustré à la figure 5-56 a été créé avec Win'Design selon les exigences du modèle 5-18.

FIGURE 5-56 **Réservation auprès d'une société aérienne modélisée avec Win'Design**



Si l'association de degré supérieur doit comporter une entité d'association, l'association de degré supérieur sera d'abord créée puis ensuite l'entité d'association. En dernier lieu, l'entité d'association sera rattaché à l'association avec l'outil *Classe association* comme pour une entité d'association sur une association binaire. La figure 5-57 montre un modèle de ce type réalisé avec Win'Design.

FIGURE 5-57 **Modèle 5-20 réalisé avec Win'Design**



Génération d'un modèle logique de données avec Win'Design

Si le modélisateur a pris soin de doter chaque attribut d'un type de données et de contraintes d'intégrité sémantique appropriées, il est possible de générer un modèle logique de données à partir d'un modèle conceptuel qui respecte très fidèlement les règles théoriques de dérivation.

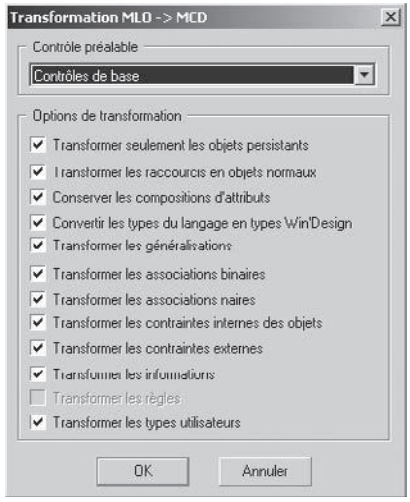
Le modélisateur doit s'assurer que chaque entité possède un identifiant explicite, sauf pour une entité d'association ou une entité agissant comme sous-type dans une association d'héritage ou encore agissant comme composant dans une association de composition. Dans ces trois derniers cas l'identifiant est implicite. On se souvient que la composition des clés primaires des tables dérivées dans le modèle logique repose sur la présence des identifiants explicites dans le modèle conceptuel.

Pour dériver un modèle logique, le modèle conceptuel doit être présent à l'écran. Le modélisateur peut alors lancer la dérivation grâce à la commande :

`Modèle->Générer modèle logique`

Win'Design procède en deux étapes et permet à chaque étape de sélectionner des options de génération. La première étape consiste à traduire le modèle conceptuel à partir de la notation UML pour produire un modèle équivalent avec la notation Merise/2. Cette opération est totalement transparente à l'utilisateur et les options offertes devraient toutes être sélectionnées (ce qui est fait par défaut) comme l'illustre la figure 5-58.

FIGURE 5-58 **Options de transformation à la première étape**



La deuxième étape consiste à générer pour un SGBD cible, le modèle logique adapté aux types de données du système cible. Elle est lancée en cliquant sur le bouton *OK* de la fenêtre précédente.

Les options générales prédéfinies sont toutes pertinentes. Le choix d'un SGBD cible devra être effectué et l'utilisateur établira s'il s'agit d'un nouveau modèle ou s'il s'agit d'une mise à jour d'un modèle existant comme le montre la figure 5-59.

Le modèle résultant est de type MLR (Modèle Logique Relationnel) et il porte par défaut le même nom que le modèle conceptuel d'origine. Il serait approprié de changer son nom pour *Modèle logique de données*.

Le modèle logique dérivé a été ajouté à l'espace de travail (Figure 5-60). Sa présentation graphique fait ressortir une singularité de Win'Design : il n'est pas possible de voir directement si un champ dans une table est à la fois clé primaire et clé étrangère. Une clé étrangère, qui ne constitue pas la clé primaire, est précédée de l'icône . **Numéro_membre** de la table **Inscription**

FIGURE 5-59 **Options de transformation à la deuxième étape**

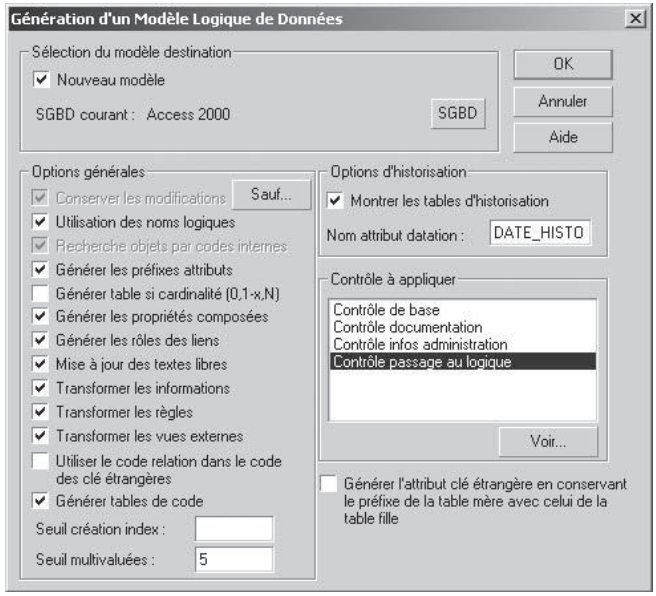
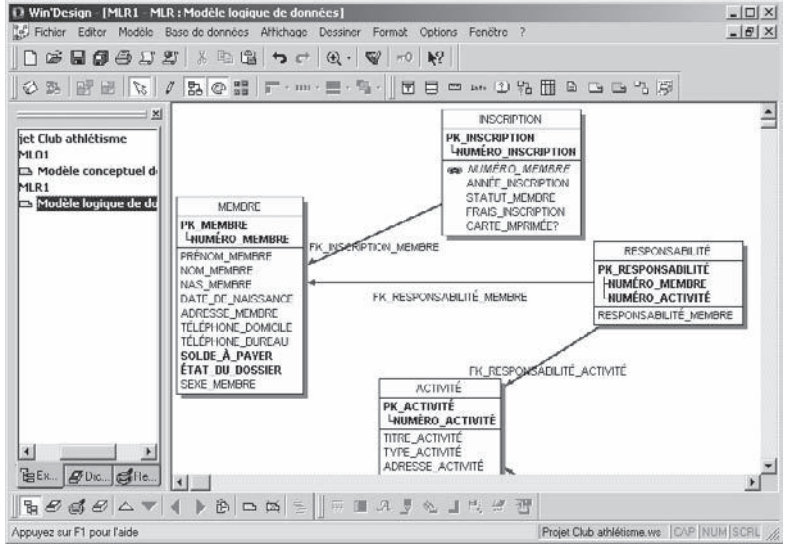


FIGURE 5-60 **Modèle logique dérivé de type MLR**



en est une. Une clé primaire simple ou composée est précédée de la mention **PK_nom table**. Ainsi la clé primaire de **Membre**, **Numéro membre** est précédée de **PK_MEMBRE-**. Néanmoins, un champ comme **Numéro_Activité** qui est à la fois un élément de la clé primaire de la table **Responsabilité** et une clé étrangère, ne comporte aucun indice visuel permettant de voir qu'il s'agit d'une clé étrangère.

C'est par la présence du lien avec la table **Membre** et par la présence dans la table **Responsabilité** de la clé primaire de la première table qu'on déduit implicitement que **Numéro_membre** est bel et bien une clé étrangère.

Les noms des liens présents dans le modèle peuvent être masqués en procédant à une sélection multiple sur ces liens (on clique sur chaque lien en gardant la touche Majuscule pressée), puis en exécutant :

```
Options->Options graphiques->de la sélection
```

Dans la fenêtre qui s'ouvre alors, on retire l'option **Afficher** à la suite de la propriété **Nom**.

Ajustements mineurs au modèle logique

Nous utilisons à nouveau le modèle conceptuel introduit à la figure 5-26. Il a été réalisé avec Win'Design et le résultat obtenu est présenté à la figure 5-61.

Le modèle logique dérivé est très fidèle à la théorie quant à la création des tables, des clés primaires et des clés étrangères. L'intégrité référentielle en ajout, en mise à jour et en suppression est appliquée judicieusement. Les associations binaires *un à un* et les associations de degré supérieur sont adéquatement prises en charge. Seul accroc à la théorie, les associations d'héritage produisent des champs redondants dans la table dérivée du super-type et dans celle dérivée du sous-type. Il y a donc lieu d'éliminer une telle redondance.

La figure 5-62 fait voir le modèle logique dérivé où on retrouve les champs **Type_activité, Titre_activité** et **Adresse_Activité** dans les tables **Activité** et **Activité de financement**. Le modélisateur devra supprimer dans les tables **Activité de financement** et **Compétition** les champs redondants **Type_activité**, **Titre_activité** et **Adresse_Activité**.

D'autres ajustements mineurs pourraient être souhaitables. Le modélisateur peut vouloir appliquer à une clé primaire de type entier un type de données à génération automatique de valeurs. La chose est possible en double-cliquant sur la table la sélection du champ dont le type doit être modifié.

FIGURE 5-61 **Modèle conceptuel réalisé avec Win'Design**

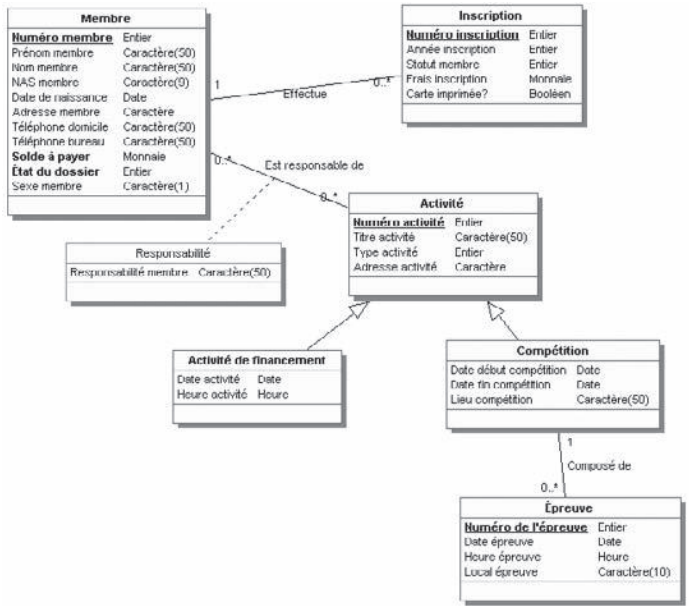
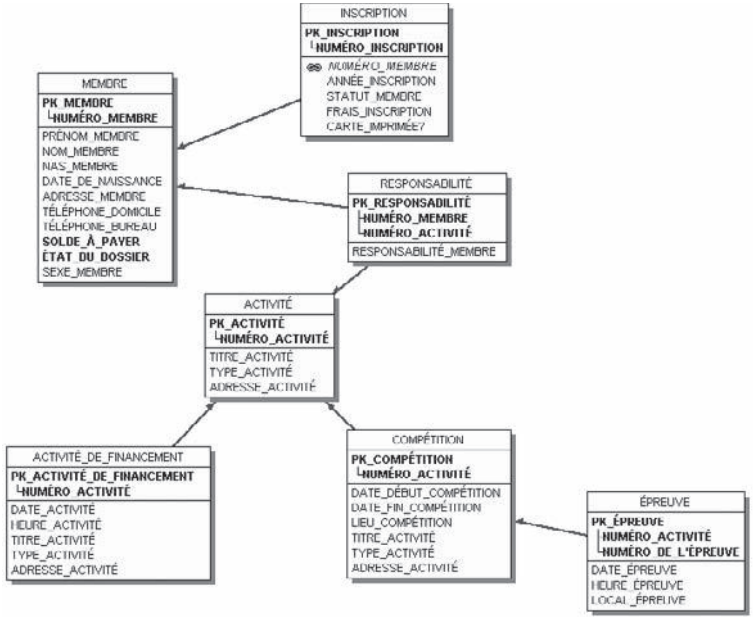


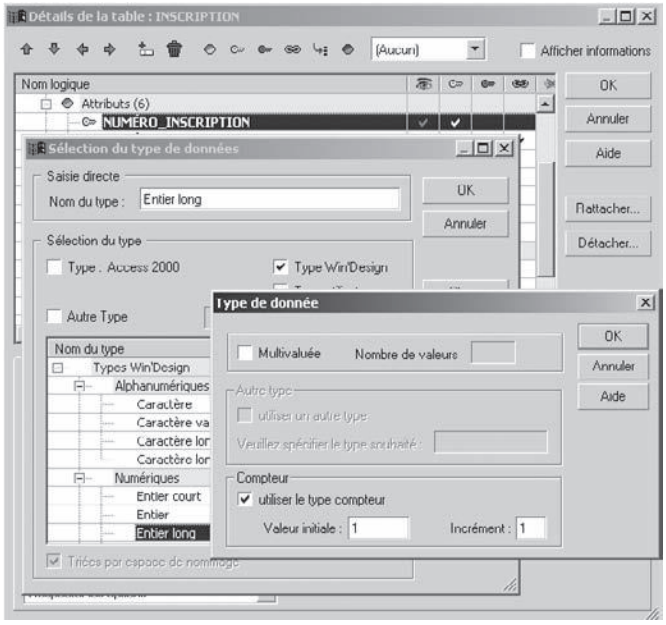
FIGURE 5-62 **Modèle logique dérivé de 5-61 avec Win'Design**



Le bouton *Type…* affiche les types Win'Design dont *Entier long* qui devra être sélectionné et le bouton *Plus…* qui permet de choisir le type *compteur* comme illustré à la figure 5-63.

Le type de données résultant appelé *Compteur(N)* génère un type *Numéroauto* en MS Access par exemple.

FIGURE 5-63 **Choix du type Compteur pour une clé primaire simple**



Nous avons mené des essais avec des modèles conceptuels comportant des associations de tous les types. Les seuls ajustements nécessaires pour être en accord à la théorie concernent le traitement des champs redondants introduits par une association d'héritage, ce qui est une performance remarquable en comparaison d'autres outils du même type.

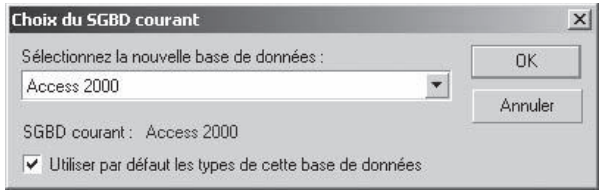
Génération du modèle physique et création de la BD

Un modèle physique est dérivé pour un modèle logique lorsque ce dernier est actif à l'écran. On s'assure d'abord que le SGBD courant est celui qui agit comme SGBD cible :

`Base de données->Sélectionner le SGBD courant…`

La fenêtre donnée à la figure 5-64 permet de fixer ce choix et d'assurer la conversion des types de données utilisés dans le modèle logique.

FIGURE 5-64 **Choix du SGBD préalable à la génération du script pour produire le modèle physique**

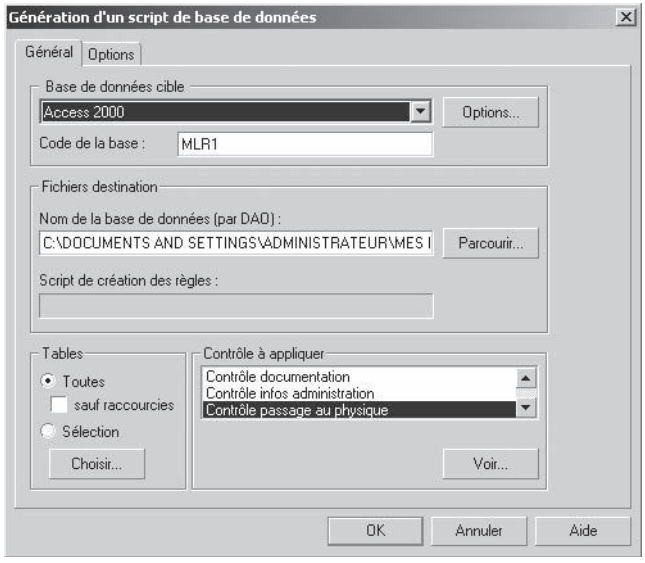


Le modélisateur peut alors procéder à la génération du script et de la BD.

`Base de données->Générer script…`

Tous les paramètres de génération du script sont fixés par défaut. Il ne reste à l'utilisateur qu'à choisir le répertoire de destination et le nom de la BD (Figure 5-65, bouton *Parcourir…)*. Si la BD existe déjà, elle sera supprimée avant la génération du script.

FIGURE 5-65 **Fenêtre assurant la génération du script et de la BD**



Aucune autre intervention n'est nécessaire. Win'Design n'affiche pas le script généré, mais une trace du processus de production du script. Si le processus devait s'arrêter pour cause d'erreur, l'utilisateur serait en mesure de noter sur quelle table ou sur quelle propriété de la table le processus a achoppé.

Le bilan

Win'Design se distingue par un respect rigoureux des règles de passage d'un modèle à un autre, telles que suggérées par la théorie, ainsi que par sa facilité d'utilisation. Il s'agit probablement d'un des outils les mieux adaptés au modélisateur débutant, car la plupart des options et des préférences établies par défaut s'accordent aux besoins de la grande majorité des modèles. Il n'en demeure pas moins un outil bien adapté au modélisateur expert et n'impose que peu de contraintes à l'utilisateur.

La prise en charge des associations de degré supérieur et la possibilité d'exprimer des contraintes inter-association, quoique sur le plan graphique seulement, en fait un des meilleurs outils de l'industrie disponibles en français pour la modélisation conceptuelle des données avec la notation UML.

Les outils de rétroconception (extraction d'un modèle logique de BD, production d'un modèle conceptuel en UML à partir d'un modèle logique) fonctionnent correctement. Les essais que nous avons menés ont produit les résultats attendus.

Il est loisible au modélisateur d'ajouter à un espace de travail un *diagramme de cas d'utilisation*, de manière à constituer un dossier d'analyse des besoins. Il s'agit essentiellement d'un outil graphique. Aucun support particulier n'est offert pour la rédaction des scénarios associés à un cas d'utilisation. Les scénarios sont inscrits textuellement, sans format établi, sous l'onglet *Contrainte* dans la fenêtre des propriétés du cas.

EN GUISE DE CONCLUSION

Ce dernier chapitre portant sur les outils de modélisation des données en UML ne se veut pas exhaustif. Nous avons retenu et évalué deux logiciels sur la base de critères précis, dont la disponibilité du logiciel en version française. Le but de l'exercice est d'exposer concrètement comment ces outils peuvent assister efficacement le modélisateur dans son travail et le libérer de tâches fastidieuses.

La modélisation des données est un exercice complexe et déroutant notamment pour le non initié. Le support d'outils appropriés est un facteur déterminant pour mener avec succès un projet de conception de bases de données. Nous prétendons que le modélisateur doit mettre tout son talent à réaliser un bon modèle conceptuel et miser sur des outils appropriés pour le libérer des arcanes de la dérivation des autres modèles. Nous espérons avoir convaincu le lecteur de la pertinence de cette thèse.

Références

[ANS 92] American National Standards Institute, ANSI Study Group on DBMS, 1992 Report.

[BJR 00] G. Booch *et al.*, Le guide de l'utilisateur UML, Eyrolles, Paris, 2000

[CHE 76] P. Chen, The Entity-Relationship Model, Towards a Unified View of Data, ACM Transactions on Database Systems, vol. 1 n° 1, 1976.

[COD 70] E.F. Codd , A Relational Model for Large Shared Data Banks, Communications of the ACM, Vol. 13, No 6, 1970.

[ConB 05] T. Connolly et C. Begg, Systèmes de bases de données, Les Éditions Raynald Goulet, Montréal, 2005.

[FOW 03] Martin Fowler, UML 2.0, CampusPress, Paris, 2003.

[GOD 03] R. Godin, Systèmes de gestion de bases de données par l'exemple, Loze-Dion éditeurs, Montréal, 2003.

[ISO 92] International Organization for Standardization, Information technology – Database languages – SQL, 1992.

[JBR 99] I. Jacobson *et al.*, Processus unifié de développement logiciel, Eyrolles, Paris, 2000.

[LAR 05] C. Larman , UML2 et les Design Patterns, 3e édition, Pearson Education, Toronto, 2005.

[MulG 00] P.A. Muller et N. Gaertner, Modélisation objet avec UML, Eyrolles, Paris, 2e édition, 2000.

[OMG 03] Object Management Group, UML 2.0 Infrastructure Specification, 2003, <www.omg.org>.

[PAS 90] D. Pascot, DATARUN : documents Données vitales et modélisation conceptuelle des données en format PDF, 1990.

[PRA 01] P.J. Prat, Initiation à SQL, Eyrolles, Paris, 2001.

[RamN 04] R. Elmasri et S. Navathe, Conception et architecture des bases de données, Pearson Education, Toronto, 2004.

[SOU 02] C. Soutou, De UML à SQL : Conception de bases de données, Eyrolles, Paris, 2002.

[SOW 84] J.-F. Sowa, Conceptual Structures- Information Processing in Mind and Machine, Addison-Wesley, New York, 1984.

[TAR 83] H. Tardieu *et al*, La méthode Merise : Principes et outils, Tome 1, Les Éditions d'Organisation, Paris, 1983.

Index

T

U-V

S'il existe de nombreux ouvrages traitant de conception de bases de données, bien peu mettent l'accent sur les modèles qui doivent être réalisés en amont. *Conception de bases de données avec UML* vise à combler cette lacune en accordant une importance prédominante au modèle conceptuel de données.

De l'analyse à la conception, cet ouvrage propose des règles, techniques, astuces et mises en garde illustrées par de nombreux exemples et études de cas qui adoptent la notation UML. Selon une démarche simple d'analyse et de conception d'une application de base de données, il vise à intégrer les techniques de modélisation et les règles de dérivation à l'intérieur d'un continuum logique et conforme à la réalité. Il présente aux modélisateurs francophones des outils logiciels leur assurant le soutien nécessaire au succès d'un projet de conception de base de données et à la cohérence des modèles.

GILLES ROY *est titulaire d'un Ph. D. en informatique de l'Université de Montréal. Depuis plus de 20 ans, il enseigne le génie logiciel et la conception des systèmes d'information. Il s'intéresse plus particulièrement à la modélisation des données et aux aspects didactiques de cette discipline. Il est professeur à l'Université du Québec à Rimouski, campus de Lévis.*

9 782760 515000
ISBN 978-2-7605-1500-0

www.puq.ca