

# Petit cours de Modélisation

## Introduction à Merise avec exercices et corrigés

- 1. Cadre général
- 2. Premiers problèmes...
- 3. Entités et relations
- 4. La normalisation
- 5. Techniques avancées

Enseigné dans le [M2 PISE](#) de l'Université Paris 7 par [Christophe DARMANGEAT](#).

Toutes les erreurs, insuffisances et plaisanteries affligeantes contenues dans ce site relèvent de la seule responsabilité de la bassesse de leur auteur (NB : attention, jeu de mots).

Si ce cours vous a instruit, ou amusé, ou mieux, les deux à la fois, n'hésitez surtout pas à me remercier en m'invitant au restaurant ou sur une barrière de corail. Ou mieux, les deux à la fois.

**Précision liminaire :** *la modélisation n'a pas de rapport direct avec la confection laborieuse de répliques miniatures de bateaux qu'on se demande toujours comment ils ont fait pour les faire entrer dans la bouteille. Ca, c'est le modélisme. La modélisation, c'est une affaire de gens sérieux, qui emploient à longueur de temps un jargon compris d'eux seuls, et qui, parfois, résolvent certains problèmes en parvenant à ne pas en créer de nouveaux.*

*Seul point commun, dans un cas comme dans l'autre, les bouteilles finissent plus souvent vides que pleines.*

## 1 – Modéliser, kesako ?

Commençons par un scoop : un système d'information a pour objectif... de traiter des informations. Autrement dit, son action va consister à accéder à des informations (qui auront donc été plus ou moins convenablement rangées ), puis à les afficher et/ou à les modifier, avant de les re-ranger pour une utilisation ultérieure. Concevoir un système d'information, c'est donc concevoir tout à la fois :

- la manière dont les données vont être **organisées** : quelles informations doivent-elles être regroupées ou, au contraire, séparées ? Comment une information permettra-t-elle d'en retrouver une autre ? Etc.
- les **traitements** proprement dits : quelles données doit-on aller chercher à quel moment, et comment doivent-elles éventuellement être modifiées ?

Selon les tâches à effectuer, un système d'information mettra davantage l'accent sur l'un ou l'autre aspect. Un système d'information à visées scientifiques, par exemple, fera sans doute beaucoup de calculs compliqués à partir d'un nombre de données pas forcément très grand. C'est donc la partie traitements qui sera la plus importante. Inversement, la consultation d'un catalogue de pièces détachées ou d'un annuaire nécessitera un minimum de traitements, mais la qualité du rangement des données y sera cruciale.

Il n'existe pas une frontière étanche entre les deux aspects : traiter les données peut inclure des opérations d'extraction et de rangement. Inversement, on ne peut se préoccuper d'organiser ses informations sans se poser un miminum de questions sur la manière d'y accéder, de les supprimer ou de les modifier. Même si elle donc pas absolue, la distinction est cependant pertinente, et permet de définir deux grands domaines de l'informatique :

1. le monde des traitements est celui de la **programmation**. L'approche de la programmation à un niveau abstrait, consistant à en formuler la logique s'appelle l'algorithmique (un mot terrifiant qui recouvre une réalité bien plus terrifiante encore, comme vous pouvez vous en rendre compte en étudiant attentivement [le cours que voici que voilà](#)).
2. le monde de l'organisation des données est celui des **bases de données** (même si toutes les données d'un système d'information ne sont pas forcément regroupées dans une « base de données », les bases de données rassemblent les informations dès lors que celles-ci se présentent en masse). L'approche abstraite de l'organisation des données dans une base s'appelle la modélisation.

La modélisation est donc à l'organisation des données ce que l'algorithmque est à leur traitement : l'approche la plus abstraite, le plan à partir duquel sera édifié le bâtiment réel.



# Petit cours de Modélisation

Introduction à Merise avec exercices et corrigés

- 1. Cadre général
- 2. Premiers problèmes...
- 3. Entités et relations
- 4. La normalisation
- 5. Techniques avancées

Enseigné dans le M2 PISE de l'Université Paris 7 par Christophe DARMANGEAT.

Toutes les erreurs, insuffisances et plaisanteries affligeantes contenues dans ce site relèvent de la seule responsabilité de la bassesse de leur auteur (NB : attention, jeu de mots).

Si ce cours vous a instruit, ou amusé, ou mieux, les deux à la fois, n'hésitez surtout pas à me remercier en m'invitant au restaurant ou sur une barrière de corail. Ou mieux, les deux à la fois.

## 2 – Premiers problèmes... et premières solutions

Organiser des données, où est le problème, direz-vous ? Y a-t-il vraiment besoin de faire des études pour cela ? Il suffit d'être un peu soigneux, et il n'y a aucune raison que ça se passe mal.

Eh bien si, en fait. Les données, quand elles se présentent en (très) grandes quantités, posent des problèmes auxquels on ne s'attend pas. C'est bien pour cela qu'on a mis au point un certain nombre de règles et de méthodes qui, si elles ne peuvent dispenser personne de réfléchir, aident néanmoins à réfléchir én évitant les catastrophes les plus courantes.

Voyons les problèmes les plus évidents qui se posent lorsqu'on veut organiser des informations. Pour cela, imaginons que nous voulions informatiser les rayons de ma discothèque... – enfin, disons les dix premiers CD du rayonnage, parce que sinon, ça va être un peu longuet.

Faisons simple pour commencer : nous ne noterons, pour chaque CD, que le titre, l'année, le nom de l'artiste et le genre musical. Cela nous donne la liste suivante :

- Nursery Cryme, Genesis, 1972, rock progressif
- Foxtrot, Genesis, 1972, rock progressif
- Selling England by the Pound, Genesis, 1973, rock progressif
- Symphonie n°2, Sibelius, 1985, classique
- Symphonie n°7, Sibelius, 1987, classique
- Concerto pour violon, Mendelssohn, 1992, classique
- Crime passionnel, Guidoni, 1982, chanson française
- 5th Gear, Brad Paisley, 2007, country
- Thick as a Brick, Jethro Tull, 1973, rock progressif
- Purpendicular, Deep Purple, 1996, rock

Il va de soi que pour représenter de telles informations, une liste c'est bien, mais qu'un tableau, ce serait beaucoup mieux. Les conventions étant en accord avec ce que nous suggère l'intuition, on mettra en ligne, les uns en-dessous des autres, les différents disques (autrement dit, en quelque sorte, les « individus » de notre base). Et on portera en colonne les différents renseignements dont on dispose pour chacun de ces individus. Naturellement, il est préférable — et, en réalité, obligatoire - de nommer ces colonnes : Ma discothèque prendrait ainsi la forme du tableau suivant :

TITRE	ARTISTE	ANNEE	GENRE
Nursery Cryme	Genesis	1972	rock progressif
Foxtrot	Genesis	1972	rock progresif
Selling England by the Pound	Genesis	1973	rock progressif
Symphonie n°2	Jean Sibelius	1985	classique
Symphonie n°7	Jean Sibelius	1987	classique
Concerto pour violon	Felix Mendelssohn	1992	Classique
Crime passionnel	Jean Guidoni	1982	chanson française
5th Gear	Brad Paisley	2007	country
Thick as a Brick	Jethro Tull	1973	rock progressif

Or, ce petit tableau, à lui seul, fait apparaître au moins deux problèmes majeurs.

- le premier, c'est que des informations identiques s'y répètent : en l'occurrence, celles concernant les artistes, et encore plus, le genre. En termes savants, on parle de **redondance** des informations. Imaginons que ma discothèque comporte plusieurs centaines de CD, il y a fort à parier que la mention « classique » ou « rock progressif » va se retrouver répliquée des dizaines de fois. Or, tout cela, il va bien falloir le stocker quelque part sous forme de bits et d'octets, et des informations inutilement répétées, ce sont des **octets inutilement occupés...** On n'en meurt pas, direz-vous. Certes, encore que.
- le second problème, lié au premier, est pour sa part rédhibitoire. C'est que vu la manière dont j'ai bâti mes informations, je ne suis pas à l'abri d'une erreur, ou même d'un simple manque d'homogénéité, dans la saisie. Par exemple, lorsque j'ai tapé le genre du CD Foxtrot, j'ai oublié les deux « s » de « progressif », ce qui est une faute de frappe assez clasique. De même, le genre « classique » est orthographié tantôt avec une majuscule, tantôt sans majuscule. Tout cela est fort préjudiciable pour la suite des événements. Si je fais une recherche, par exemple, sur le genre « rock progressif » ou « classique », il manquera des CDs qui auraient dû y figurer. Cette organisation laisse donc la possibilité qu'existe une **hétérogénéité** des données, un des pires cauchemars de l'informaticien (qui pourtant n'en manque pas) .

Ces deux problèmes possèdent une solution commune, très simple mais extrêmement efficace, qui consiste à recenser **séparément** les CD et les genres :

TITRE	ARTISTE	ANNEE
Nursery Cryme	Genesis	1972
Foxtrot	Genesis	1972
Selling England by the Pound	Genesis	1973
Symphonie n°2	Jean Sibelius	1985
Symphonie n°7	Jean Sibelius	1987
Concerto pour violon	Felix Mendelssohn	1992
Crime passionnel	Jean Guidoni	1982
5th Gear	Brad Paisley	2007
Thick as a Brick	Jethro Tull	1973

GENRE
chanson française
classique
country
rock progressif

Il ne reste plus à présent qu'à préciser à quel genre appartient chaque CD. Pour ce faire, on identifie chaque genre par un code unique, et on reporte ce code dans le tableau des CDs.

CODE	GENRE
1	chanson française
2	classique
3	country
4	rock progressif

avec :

TITRE	ARTISTE	ANNEE	CODE
Nursery Cryme	Genesis	1972	4
Foxtrot	Genesis	1972	4
Selling England by the Pound	Genesis	1973	4
Symphonie n°2	Jean Sibelius	1985	2
Symphonie n°7	Jean Sibelius	1987	2
Concerto pour violon	Felix Mendelssohn	1992	2
Crime passionnel	Jean Guidoni	1982	1
5th Gear	Brad Paisley	2007	3
Thick as a Brick	Jethro Tull	1973	4

Comme Bertin, nous venons de faire d'une pierre deux coups :

- on a **économisé de la place en mémoire**, car stocker un code mobilise beaucoup moins d'octets que stocker un intitulé
- surtout, on a pris **une garantie contre les informations hétérogènes** : le code correspond toujours au même intitulé, et on ne peut plus guère imaginer se retrouver avec un même genre musical orthographié différemment.

Ce que nous venons de faire, c'est – en tout petit, petit, le fond de la question en matière de modélisation de l'information : nous venons de créer une **relation** entre deux tables (deux entités) ; voilà pourquoi on parle de **Systèmes de Bases de Données Relationnelles**. Tout ce que nous allons voir ensuite, n'est pour ainsi dire que des complications et des raffinements à partir de cette base simple.

Cela dit, jusqu'à maintenant, nous avons procédé pour ainsi dire uniquement par intuition – or, vous vous en doutez bien, il existe un certain nombre de formes pré-établies, de standards (tant de méthode que de représentation) pour venir à bout des problèmes les plus complexes. Ce sont ces formes et ces standards que nous allons aborder à présent.

**Remarque capitale** : Il suffit d'observer la situation à laquelle nous sommes parvenus pour constater que nous n'avons parcouru que la moitié du chemin. Nous avons certes éliminé une source de redondances en créant la table Genres. Mais nous en avons laissé une deuxième : celle liée aux artistes. Il est donc essentiel de comprendre que ce qui est présenté ici n'est que le premier pas, et que notre modélisation est irrecevable en l'état.





# Petit cours de Modélisation

Introduction à Merise avec exercices et corrigés

1. Cadre général
2. Premiers problèmes...
3. Entités et relations
4. La normalisation
5. Techniques avancées

Enseigné dans le [M2 PISE](#) de l'Université Paris 7 par [Christophe DARMANGEAT](#).

Toutes les erreurs, insuffisances et plaisanteries affligées contenues dans ce site relèvent de la seule responsabilité de la bassesse de leur auteur (NB : attention, jeu de mots).

Si ce cours vous a instruit, ou amusé, ou mieux, les deux à la fois, n'hésitez surtout pas à me remercier en m'invitant au restaurant ou sur une barrière de corail. Ou mieux, les deux à la fois.

## 3 – Merise et le diagramme « entités-relations »

*Merise est le nom d'une méthode, ou d'un ensemble de méthodes, développée en France dans les années 1970, et qui a été très largement employée depuis lors. Depuis une quinzaine d'années, Merise laisse peu à peu place à [UML](#), une autre norme nous n'aborderons pas ici (mais vous aurez tout de même bien du mal à y échapper).*

*Les puristes qui tomberont sur ce cours seront sans doute horrifiés devant la manière dont il taille dans le vif du sujet : loin de faire ici une véritable présentation de Merise, il s'agit simplement, si l'on veut, d'une introduction au début du commencement d'une esquisse d'une approche de Merise.*

*L'auteur de ces lignes revendique pleinement ses choix. D'abord, parce que ce cours s'adresse à de complets débutants en informatique, et qu'il faut bien commencer par quelque part. Ensuite, parce que le volume horaire dudit cours ne permet pas d'aborder autre chose que les grands fondamentaux – et comme dans tout apprentissage raisonné, mieux vaut moins, mais mieux. Enfin, parce que le propos de ce cours est bien moins de faire ingurgiter une série de conventions plus ou moins arbitraires que de permettre à ceux qui le suivront de s'approprier les raisonnements qui se cachent derrière.*

### 3.1 – Le cadre général

Merise constitue donc un ensemble très riche de méthodes et de représentations, dont nous ne verrons ici qu'une petite partie - mais la plus cruciale.

Toute base de données va donner lieu à une **double représentation** :

1. le plan le plus abstrait (mais qui contient déjà toutes les informations indispensables pour la construction de la base de données) . Ce plan porte le nom de fort poétique de [Modèle Conceptuel de Données \(MCD\)](#).
2. un plan plus proche de ce que sera la base effective, telle qu'elle sera réalisée sur machine : le [Modèle Logique des Données \(MLD\)](#).

Le point crucial à enregistrer dès maintenant, c'est que le **MLD se déduit strictement du MCD d'après des règles formelles**. Autrement dit, une fois le MCD réalisé, il n'y a plus besoin de réfléchir une seule seconde pour produire le MLD : tout se fait par automatismes. La meilleure preuve, c'est qu'il existe des logiciels qui se proposent de réaliser le MLD d'un clic de souris, d'après le MCD. En revanche, il n'existe rien de tel pour concevoir le MCD : le seul ingrédient qui entre dans sa composition est l'huile de neurones. N'oubliez pas de faire quelques provisions...



### 3.2 – Le MCD, les entités et les relations

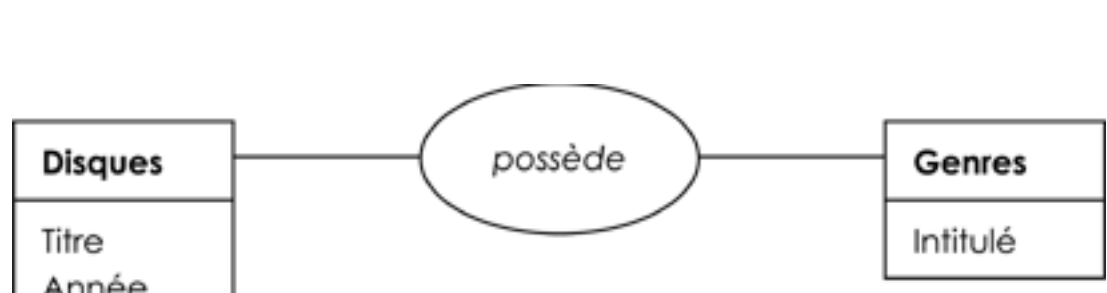
Les informations à traiter doivent être regroupées en ensembles cohérents, comme dans les tableaux que nous avons constitués il y a un instant. Dans les conventions de Merise, ces ensembles s'appellent des **entités**, et sont symbolisés par des **rectangles**. Chaque entité porte un **nom**, qui l'identifie de manière unique. Ce nom sera obligatoirement un **substantif au pluriel** : pour notre discothèque, on propose très logiquement « **Disques** » et « **Genres** ».

Les entités comprennent toujours un certain nombre d'éléments appelés **propriétés** (on parlera aussi d'**attributs**). Il s'agit des différentes rubriques qui devront être renseignées pour chaque individu.

Chaque entité, lorsqu'on passera au MLD (puis à la réalisation concrète de la base) donnera lieu à un tableau (on parle plus volontiers de **tables**). L'entité Disques du MCD produira donc une table Disques dans le MLD, et l'entité Genres, une table Genre. Les différentes propriétés de l'entité, qui sont donc écrites les unes sous les autres, deviendront **les titres des colonnes** de ces tables. Et dans ces colonnes, on fera figurer les différentes valeurs que prennent ces propriétés pour chacun des éléments de nos tables. Si vous trouvez cette explication un peu compliquée, pensez tout simplement à l'entité / table Disques : le titre, l'année et l'artiste sont disposés les uns sous les autres lorsqu'on parle de l'entité, et les uns à côté des autres (ce sont des en-têtes de colonne) lorsqu'on la représente comme une table.

Il ne reste plus à signifier que pour que chaque CD possède un genre (et pas n'importe lequel), mes deux entités doivent se trouver en **relation** l'une avec l'autre. Cette relation (on peut aussi parler d'**association**) sera symbolisée par un **ovale**, et sera nommée (par un **verbe**).

Voilà donc ce que cela donne :



Cette représentation ne se lit pas n'importe comment. Pour être certain de ne pas commettre de contresens, lorsqu'on traduit le schéma ci-dessus, il vaut mieux éviter de dire « *les disques possèdent des genres* », ou pire encore « *la table disque possède certains genres* ». La bonne traduction, celle qui vous évitera au maximum de commettre des erreurs, consiste à dire que « **Chaque élément de la table Disques possède un (ou plusieurs ?) genres** ». En prenant l'affaire par l'autre bout, on peut tout aussi bien dire (même si c'est un peu laid à l'oreille) : « Chaque genre est possédé par un (ou plusieurs ?) disques » (on verra un peu plus loin comment en avoir le cœur net sur ces points d'interrogation).

Résumons-nous :

- les données doivent être systématiquement regroupées de manière à éviter les **redondances**, source de gâchis et, surtout, d'erreurs. Ce regroupement est la base de la modélisation.
- chacun des groupements (correspond, dans le MCD, à une **entité**, qui se traduira par une table dans la Base de Données.
- toute **entité** est symbolisée par un **rectangle**. Elle est nommée par un substantif au pluriel, désignant les éléments qu'elle contient
- les **propriétés** d'une **entité** correspondent aux colonnes de la table qui sera déduite de cette entité.
- les entités (c'est-à-dire : les individus présents dans les entités) peuvent être mises en rapport via des **relations** (ou **associations**)
- toute **relation** est symbolisée par un **ovale**. Elle est nommée par un verbe.

**Remarque capitale** : l'étape consistant à créer autant d'entités que nécessaire est la première de toutes, et elle est absolument fondamentale. Le critère est simple : **aucune valeur de propriété ne doit se répéter** (sauf peut-être, à titre exceptionnel) **dans aucune table**. Cette règle, intangible quand il s'agit de valeurs de type texte, peut être assouplie lorsqu'il s'agit de valeurs numériques (dont les dates) : il serait en effet un peu ballot d'économiser la répétition d'un nombre... au prix de la création d'un code permettant d'accéder à ce nombre : ce serait payer de la main gauche ce qu'on économise de la main droite. Cependant, créer une entité composée uniquement de nombres peut se justifier, lorsqu'on veut restreindre les possibilités à un certain ensemble de valeurs (par exemple, les différentes motorisations disponibles pour une automobile).



### 3.3 – Éléments supplémentaires

#### 3.3.1 – Deux nouveaux mots de vocabulaire

Dans une table, on évite de parler de « lignes » et de « colonnes ».

- Les lignes correspondent aux différents individus, ou aux différents objets individuels, répertoriés dans une table : dans la table Disques, chaque ligne correspond à un de mes CD. Ces différents éléments individuels qui correspondent aux lignes sont appelés **enregistrements**.
- Les colonnes, qui correspondent aux propriétés de l'entité dans le MCD, sont appelées des **champs**.

#### 3.3.2 – Typage des propriétés

Vous ne serez guère étonnés d'apprendre que les informations contenues dans les entités (donc, dans les tables) vont devoir au bout du compte être codées numériquement afin d'être stockées sur un support informatique, sous un nom qui est celui de la propriété.

Pour chaque enregistrement, celle-ci se comporte donc comme un **nom de variable**... ce qui est somme toute logique, car à de menus détails près, c'en est une. Tout ceci nous amène au fait que les propriétés, à l'instar des variables, relèvent de certains **types**.

Dans le détail, les types disponibles pour les propriétés varient légèrement d'un système de gestion de bases de données à l'autre. En ce qui nous concerne, nous pouvons en rester à un niveau assez général, en considérant les types les plus courants :

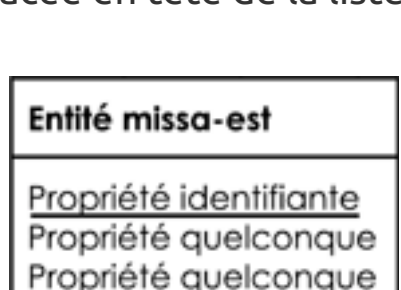
- **numérique** : on y distingue systématiquement l'entier (« Integer ») du nombre à virgule (« Float »). Le type « AutoIncrement », souvent utilisé pour gérer les **clés primaires** [MODE NO PANIC ON] vous saurez ce que c'est très bientôt [MODE NO PANIC OFF], correspond à un entier dont la valeur est automatiquement attribuée à la création d'un nouvel enregistrement. Les bases de données proposent également toujours au moins un type Date/Heure.
- **texte** : on aura éventuellement différents types correspondant à différentes longueurs maximales du texte.
- **booléen** : inévitable !

Outre les informations précédemment citées, les documents de modélisation, MCD et MLD, devront donc faire apparaître, pour chaque entité, le type de chaque propriété.



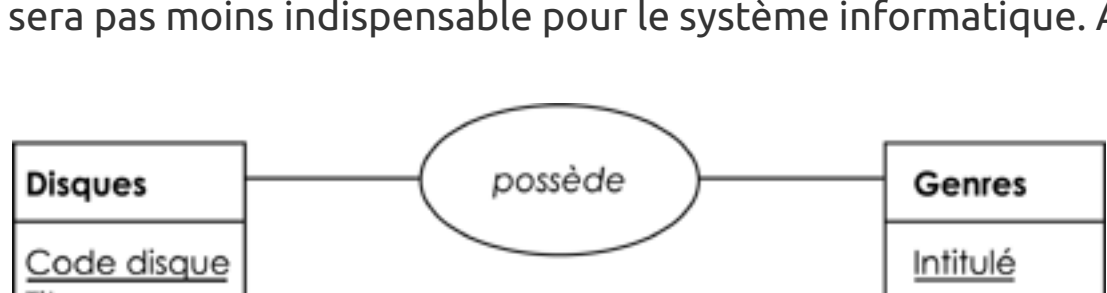
### 3.3 – Propriété identifiante (clé primaire)

Tout système de bases de données impose dans chaque entité, chaque individu (chaque enregistrement) puisse être identifié de manière unique, sans ambiguïté, par la machine. Le procédé le plus courant consiste à dédier à cela une propriété spéciale, appelée **propriété identifiante** ou encore **clé primaire**. On peut constituer une clé primaire à partir d'une combinaison de champs, mais nous verrons que c'est une solution qui n'est employée que dans certains cas particuliers ; restons-en donc pour le moment à l'idée que la clé primaire est un champ spécial. La clé primaire est alors généralement placée en tête de la liste des propriétés, en la soulignant pour indiquer son statut particulier :



Il est en fait assez rare de trouver spontanément une propriété capable de jouer ce rôle. Même les propriétés qui semblent faire de bonnes candidates (par exemple, une plaque d'immatriculation ou un numéro de sécurité sociale) ne sont pas forcément aussi opportuns qu'elles en ont l'air, pour un certain nombre de raisons. Et il n'est pas rare qu'aucune des propriétés présentes ne puisse nous prémunir contre les doublons ; c'est le cas avec l'entité *Disques* de notre exemple : plusieurs CD peuvent très bien avoir le même titre, et je ne parle pas de l'auteur ni de l'année. On ne peut pas davantage exclure la possibilité que deux auteurs homonymes aient sorti la même année un disque portant le même titre (ce qui nous empêche donc d'avoir confiance dans une clé primaire constituée de la combinaison des trois propriétés).

Voilà pourquoi le plus souvent, on sera amené à créer une **propriété supplémentaire** destinée uniquement à jouer le rôle d'identifiant / clé primaire. Il s'agira presque toujours d'un code, unique pour chaque occurrence de l'entité (et voilà pourquoi un nombre de type « autoincrément » est si pratique). Ce code sera rarement visible par l'utilisateur, qui ignorera sans doute son existence : il n'en sera pas moins indispensable pour le système informatique. Ainsi, notre modèle de discothèque deviendra-t-il :



### 3.4 – Les cardinalités

*Contrairement à ce que certains pourraient penser, ce terme n'indique ni le fait de devenir cardinal (Dieu m'en garde !) ni le fait de se situer à l'un des quatre points cardinaux (en l'occurrence, complètement à l'Ouest). Non, la cardinalité, c'est un mot savant de mathématicien pour dire tout bêtement que l'affaire a un rapport avec des nombres et des quantités. Pour dire la même chose, on aurait pu donc tout simplement parler de « quantité » ou de « numération ». Sauf que dans un diner de famille, placez subrepticement « cardinalité » entre la poire et le fromage, et vous pourrez vérifier par vous-mêmes que l'effet obtenu n'est pas du tout le même.*

#### 3.4.1 – Encore et à nouveau la discothèque

Dans le MCD que l'on vient d'élaborer, il manque une information essentielle pour la suite : à combien d'éléments de l'autre entité chaque élément peut-il être associé ? Lorsqu'on bâtit une relation entre deux entités, on doit nécessairement préciser ce point, car de lui dépendent de très importantes conséquences.

Dans l'exemple que nous avons pris, celui de la discothèque, il paraît évident qu'un même genre musical peut être représenté par plusieurs disques. En sens inverse, en revanche, on peut être embêté pour décider à combien de genres peut correspondre chaque disque. On peut en effet imaginer soit que chaque disque ne puisse être rattaché qu'à un genre et un seul afin de faciliter le classement, soit qu'à chaque disque on puisse attribuer plusieurs genres à la fois, ce qui introduit davantage de complexité, mais aussi davantage de souplesse.

La décision, dans cette alternative, n'appartient pas à l'informaticien : il s'agit d'un choix d'ordre fonctionnel, qui doit être subordonné aux besoins de l'organisation pour laquelle est conçue de la base de données. Il n'existe donc aucune règle qui permette de trancher a priori entre les deux possibilités... excepté que la technique doit être au service des besoins de ceux qui s'en serviront, et non l'inverse. En revanche, ce qui nous intéresse ici, ce sont les conséquences de ce choix sur notre base de données.

#### 3.4.2 – Cardinalités minimum et maximum

Un modèle (conceptuel), lorsqu'il met en relation deux entités A et B, doit toujours stipuler à combien d'éléments de l'entité B chaque élément de A peut correspondre, et inversement – c'est ce qu'on appelle la définition des **cardinalités**. De là, il faut distinguer le nombre **minimum** et le nombre **maximum** de ces correspondances : pour chaque élément d'une entité, on doit donc stipuler à combien d'éléments de l'autre entité celui-ci est susceptible de correspondre, au minimum et au maximum. Ainsi, toute relation entre deux entités impose de préciser quatre nombres (quatre cardinalités) : cardinalité minimum de A vers B, maximum de A vers B, minimum de B vers A, maximum de B vers A.

En reprenant l'exemple de la dicothèque, cela revient à se poser les questions suivantes :

- À combien de genres au minimum correspond chaque CD ? (autrement dit : un CD peut-il ne pas avoir de genre, ou en a-t-il forcément au moins un ?)
- À combien de genres au maximum correspond chaque CD ? (autrement dit : un CD peut-il avoir plusieurs genres, ou est-il limité à un seul ?)
- À combien de CD au minimum correspond chaque genre ? (autrement dit : ma table des genres comprend-elle uniquement des genres qui correspondent à mes CD, ou peut-il y avoir des genres « orphelins » ?)
- À combien de CD au maximum correspond chaque genre ? (autrement dit : puis-je avoir plusieurs CD du même genre, ou est-ce interdit ?)

#### 3.4.3 – Les valeurs possibles

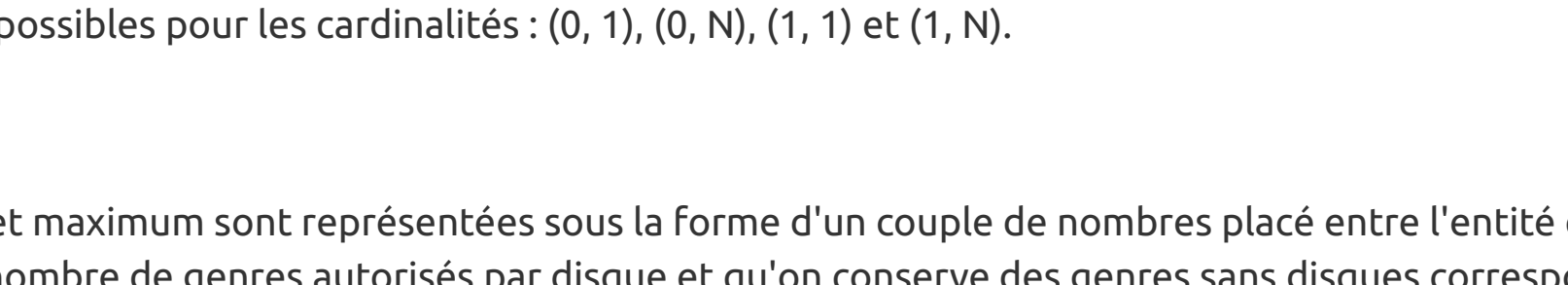
Les cardinalités obéissent à un formalisme assez étroit :

- La cardinalité minimum ne peut prendre que les valeurs 0 ou 1. Autrement dit, soit on considère qu'un élément de la table A *peut* être en relation avec un (ou plusieurs) éléments de la table B (mais que ce n'est pas obligatoire), soit on considère que tout élément de la table A doit impérativement être en relation avec au moins un élément de la table B. Dans notre exemple, choisir 1 comme cardinalité minimum signifie qu'un disque doit être classé dans au moins un genre. Choisir 0 signifie qu'on estime que certains disques n'ont pas forcément de genre (le même problème se pose dans l'autre sens de la relation, pour savoir s'il peut exister ou non des genres sans disques).
- La cardinalité maximum ne peut valoir que 1 ou N, autrement dit un ou plusieurs. C'est la discussion de tout à l'heure à propos des genres : autorise-t-on ou non chaque CD à être classé dans plusieurs genres à la fois ? En sens inverse, il ne fait aucun doute qu'à chaque genre, doivent pouvoir correspondre plusieurs disques.

Il n'existe donc que quatre cas de figure possibles pour les cardinalités : (0, 1), (0, N), (1, 1) et (1, N).

#### 3.4.4 – Représentation

Dernier point, les cardinalités minimum et maximum sont représentées sous la forme d'un couple de nombres placé entre l'entité et la relation. Par exemple, dans le cas de notre discothèque, si on limite à un seul le nombre de genres autorisés par disque et qu'on conserve des genres sans disques correspondants, on a :



#### 3.4.5 – Raccourcis de langage

Pour décrire les cardinalités, on va souvent user d'un raccourci de langage. Comme les cardinalités les plus décisives sur l'architecture de la base sont souvent les cardinalités maxima, on aura tendance à ne parler que de elles. Ainsi, dans le cas d'une relation où l'une des deux cardinalités maximales vaut 1 et l'autre N, on dira volontiers qu'on a affaire à une relation « **un à plusieurs** ». Lorsque les deux cardinalités maximales valent N, on parlera de relation « **plusieurs à plusieurs** ».

Une relation de type « un à un » (où, donc, les deux cardinalités maximum sont égales à 1) est un cas limite. Cela signifie que nous avons créé deux entités qui en réalité n'en forment qu'une seule, puisque chaque élément de l'une correspond à un élément de l'autre, et à un seul. Ce n'est pas à proprement parler une faute, mais face à une telle situation, on a toujours intérêt à se demander ce qui justifie d'avoir créé deux entités plutôt qu'une seule.

### 3.5 – Du MCD au MLD : première approche

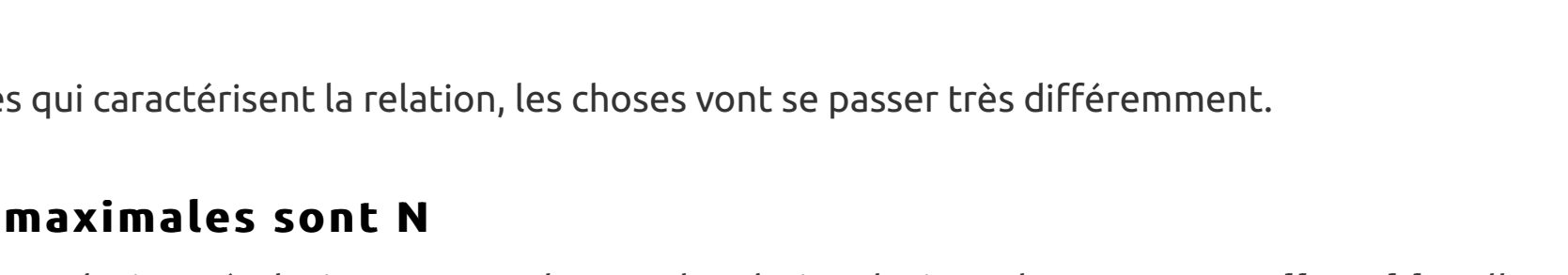
Je rappelle que par rapport au MCD, le MLD est un plan moins abstrait et plus proche de la réalité, c'est-à-dire de la base de données telle qu'elle existera sur les machines. Je rappelle aussi que toute la difficulté de la modélisation réside dans l'élaboration du MCD. Une fois que celui est conçu, le MLD s'en déduit par l'application de quelques règles (on pourrait dire : d'un algorithme). Donc, le passage du MCD au MLD n'est qu'une question de rigueur, et plus du tout d'intelligence ou d'imagination (« surtout pas ! », pourrait-on même dire).

Une première règle, d'une simplicité biblique, est que **toute entité du MCD devient une table du MLD. L'identifiant de chaque entité devient la clé primaire de chaque table**.

Ensuite, selon les cardinalités maximales qui caractérisent la relation, les choses vont se passer très différemment.

#### 3.5.1 Si les deux cardinalités maximales sont N

...autrement dit, si la relation est de type « plusieurs à plusieurs ». Dans le MLD, la relation devient alors une **nouvelle table**, elle-même en relation avec les deux tables produites par les deux entités. Une telle table est dite **table de correspondance**, ou encore **table de liaison**, **table de jonction**, **table d'association**, etc. Elle ne contient pas à proprement parler des données : son rôle est d'organiser les rapports entre les éléments des tables qui, elles, les contiennent. Une table de jonction contiendra uniquement des propriétés correspondant aux clés primaires des deux entités, qu'elle associera deux à deux :



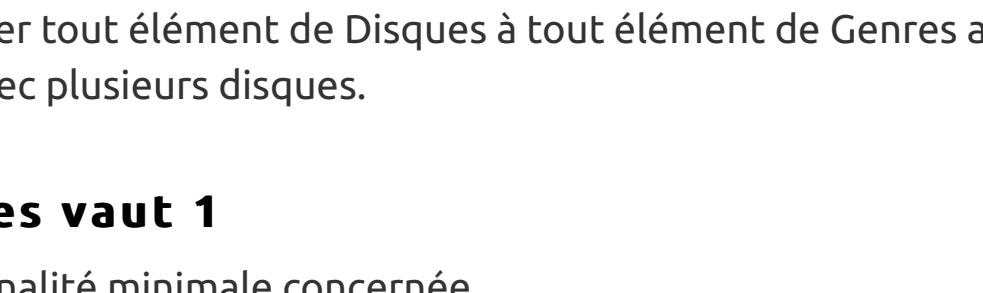
C'est très logique : la table de jonction va permettre d'associer tout élément de Disques à tout élément de Genres autant de fois que souhaité. Un même disque pourra ainsi être mis en rapport avec plusieurs genres, et un même genre avec plusieurs disques.

#### 3.5.2 Si l'une des deux cardinalités maximales vaut 1

Ce cas se divise à son tour en deux, selon la valeur de la cardinalité minimale concernée.

##### 1. On a une cardinalité (1, 1)

Concrètement, cela veut dire, dans notre exemple, que chaque CD possède un genre et un seul. Un CD ne peut pas ne pas avoir de genre ; il ne peut pas non plus en avoir plusieurs. Dans ce cas, dans le MLD, cette relation devient une relation **directe** entre les deux tables. L'identifiant de la table côté « plusieurs » devient une nouvelle propriété de la table du côté « un », ainsi que l'illustre cet exemple :



Quand on y réfléchit, c'est parfaitement logique : pour chaque disque, il faudra renseigner un code (et un seul) qui correspondra à l'un des genres présents dans la table Genres. La table Genre, elle, ne contient aucun code renvoyant vers les disques - sinon, cela voudrait dire que chaque Genre renvoie vers un disque et un seul ! En fait, on vient de retrouver par un long détour l'exemple à partir duquel on avait introduit le raisonnement (« Tout ça pour ça », direz-vous...)

Et, au passage, pour cette nouvelle propriété dans la table Disques qui contiendra une valeur prise par la clé primaire de la table Genre, on dit qu'il s'agit d'une **clé étrangère**.

##### 2. On a une cardinalité (0, 1)

**Remarque essentielle** : le raisonnement qui suit concerne en réalité **toute** situation dans laquelle la cardinalité minimale est égale à zéro. Cela correspond à la situation où chaque CD peut avoir un genre (au maximum) mais où il n'est pas obligé d'en avoir un. Là, les informaticiens se divisent en deux catégories. Il y a les couillants (les autres préféreront les appeler les laxistes), qui diront : « Faisons au plus simple. Il suffit de créer une clé étrangère ; le disque n'aura pas de genre attribué, la valeur de la clé étrangère sera vide. Après tout, il n'y a pas de mal à cela. » Au passage, une valeur vide, dans une base de données, s'appelle en jargon une valeur **Null**. A cela, l'autre catégorie d'informaticiens, beaucoup plus nombreuse, que sont les rigoureux (et que les premiers appelleront les psycho-rigides) rétorquent : « Certes, on peut avoir des valeurs Null dans une table. Mais ce n'est jamais une bonne chose. On ne sait pas, par exemple, si c'est un défaut de saisie ou une valeur volontaire. Et puis, quand on fera des recherches ou des traitements automatisés, cela risque de nous jouer de bien vilains tours. Il est donc nettement préférable de jouer la sécurité : il suffit de créer une table de jointure, exactement comme dans le cas d'une relation « plusieurs à plusieurs ». Dans ce cas, plus de Null ; en revanche, il faudra mettre en place un contrôle pour être certain que chaque disque n'apparaît pas plus d'une fois dans la table d'association...

On remarquera qu'en ce qui concerne les normes de représentation des cardinalités, c'est... la jungle. Les images ci-dessus, réalisées avec LibreOffice Base, placent en effet dans le MLD le « 1 » du côté de la clé primaire et le « plusieurs » du côté du code qui pointe vers celle-ci, soit exactement l'inverse de ce qu'impose Merise dans le MCD. Bon, d'un autre côté, comme Merise fait pour sa part exactement le contraire d'UML, on peut dire en quelque sorte que quelque part (mais où ?), on retombe sur nos pattes. Menfin, la seule vraie conclusion à tirer est que le positionnement des cardinalités relève de la pure convention, pour ne pas dire de l'arbitraire le plus total.

Eh bien avec tout ça, il est grand temps de passer à la pratique :

exercice 1

exercice 2





# Petit cours de Modélisation

Introduction à Merise avec exercices et corrigés

- 1. Cadre général
- 2. Premiers problèmes...
- 3. Entités et relations
- 4. La normalisation
- 5. Techniques avancées

Enseigné dans le [M2 PISE](#) de l'Université Paris 7 par [Christophe DARMANGEAT](#).

Toutes les erreurs, insuffisances et plaisanteries affligeantes contenues dans ce site relèvent de la seule responsabilité de la bassesse de leur auteur (NB : attention, jeu de mots).

Si ce cours vous a instruit, ou amusé, ou mieux, les deux à la fois, n'hésitez surtout pas à me remercier en m'invitant au restaurant ou sur une barrière de corail. Ou mieux, les deux à la fois.

## 4 – La normalisation du modèle

Ce vocable barbare recouvre l'idée que pour être valide, un modèle doit obéir à un certain nombre de normes. Celles-ci ne sont pas toutes du même ordre : certaines sont de simples règles de bonne conduite (ou d'hygiène), destinées à évacuer certaines ambiguïtés pouvant s'avérer gênantes à l'usage. D'autres sont impératives et leur violation constitue à coup sûr une faute grave. Le mieux est de les respecter toutes, en comprenant pourquoi il ne s'agit pas de préceptes tombés du ciel dans le seul but de nous pourrir l'existence, mais au contraire, de pratiques rationnelles visant à nous la simplifier.

### 4.1 – Les redondances

C'est la règle principale, sur laquelle on ne saurait que trop insister : **une modélisation correcte doit avoir éliminé toutes les redondances possibles de données en constituant des nouvelles entités chaque fois que nécessaire**, comme nous l'avons fait dès le départ en regroupant les genres musicaux dans une entité séparée.

On doit donc explorer systématiquement toutes les propriétés de toutes les entités, en se demandant à chaque fois si les valeurs que prendront ces propriétés ne risquent pas de se répéter plus souvent qu'à leur tour.



### 4.2 – Les valeurs multiples

Créer une propriété aux valeurs multiples constitue sans doute une des plus grosses fautes possibles dans une base de données.

Les valeurs multiples ne doivent pas être confondues avec les redondances. Par valeurs multiples, on entend le fait qu'une même propriété, pour chaque enregistrement, soit censée posséder plusieurs valeurs à la fois. Prenons l'exemple d'un journal qui souhaite mémoriser ses articles et leurs auteurs. On imagine donc une entité Articles, avec le titre, le corps de l'article, sa date, etc. ... Ayant bien assimilé la partie précédente, on prend soin de constituer une entité Auteurs et d'établir une relation entre Articles et Auteurs.

Fort bien. Sauf qu'à ce moment-là, le rédacteur en chef nous fait remarquer qu'un article peut fort bien avoir été écrit par plusieurs auteurs. « Qu'à cela ne tienne ! », répondons-nous bravement. Et de là, il y a trois solutions... dont deux qui mènent droit à la catastrophe.

- la première, que j'ose à peine concevoir, consiste à oublier tout ce que nous avons vu jusque-là et à attribuer une propriété Auteurs (au pluriel !) à notre entité Articles. C'est évidemment un grand n'importe quoi. D'une part, cette propriété viendrait tout à la fois télescoper l'entité Auteurs et la relation que nous avons établie avec elle. D'autre part, on se demande bien ce que pourrait recouvrir une propriété figurant au pluriel, et possédant donc plusieurs valeurs à la fois pour chaque enregistrement de la table...
- la deuxième est la plus fréquente, et c'est celle-là que j'ai appelée « la faute des valeurs multiples ». **Elle consiste à créer autant de propriétés que de co-auteurs possibles**, en les appelant en général Auteur1, Auteur2, Auteur3, etc., et en considérant que chacune de ces propriétés est en relation avec l'entité Auteurs. Cette architecture va produire un galimatias à peine moins horrible que le précédent : outre qu'on nage en pleine confusion entre MCD et MLD, il est évidemment hors de question d'avoir une entité ressemblant à un emmental, avec des Null à tous les coins de rue (car tous les articles n'auront pas le maximum d'auteurs que nous avons prévu).

Ces deux fautes incarnent les deux formes possibles des propriétés à valeurs multiples. Elles sont tout aussi catastrophiques l'une que l'autre.

- en fait, la troisième solution est non seulement la seule correcte, mais c'est aussi, en réalité, la plus simple : la relation entre Articles et Auteurs est de type plusieurs à plusieurs, et se traduira dans le MLD par la création d'une table de jonction.



### 4.3 – Les synonymes

Les redondances d'informations au sein d'une même propriété ne sont pas les seules à devoir être traquées. Dans un modèle un peu compliqué, on peut vite être amené, volontairement ou non, à faire figurer plusieurs fois (dans plusieurs entités différentes) la même information – mettons, le nom du titulaire d'une police d'assurance. Lorsqu'une telle situation se produit, c'est toujours – hormis dans des contextes très particuliers – une erreur.

On se fixera donc comme principe qu'**une information doit toujours figurer de manière unique dans un modèle**. En effet, outre la perte de place occasionnée par une telle répétition, c'est surtout le risque d'une divergence entre les différentes copies de l'information qui est à craindre ; par exemple, une même personne qui se retrouverait au bout d'un moment répertoriée sous des noms à l'orthographe diverse selon qu'on est en train de parler de son assurance-vie, de son logement ou de ses versements de mensualités....

Ce deuxième principe peut être d'autant plus difficile à appliquer (mais d'autant plus nécessaire) que l'entreprise ou l'organisation elle-même répertorie les mêmes informations sous des noms différents (par exemple, le « code produit » dans un service, et la « référence » dans un autre). L'informaticien peut donc être trompé par ces noms différents et ne pas comprendre qu'ils désignent en fait la même information.

Ce type de biais est appelé, dans le jargon de Merise, un synonyme. Et les synonymes doivent donc être éliminés méthodiquement et sans pitié.

**Synonymes indirects** : j'appelle ainsi une situation où plusieurs informations différentes figurent dans la base alors que certaines peuvent être déduites des autres. Imaginons par exemple que nous trouvions dans notre base une liste de marchandises avec leurs prix hors taxes, les taux de TVA, le montant de la TVA et les prix toutes taxes. Il est clair que **les deux dernières informations se déduisent des deux premières par calcul**. Donc, la mauvaise architecture choisie conduit, au mieux, à stocker des informations pour rien (si les calculs sont justes) ; soit à des discordances d'information s'ils sont faux.



### 4.4 – La polysémie

La polysémie pourrait être un nom de maladie. Dans un sens, d'ailleurs, c'en est effectivement une. Elle est l'exact inverse du synonyme, mais elle est tout aussi nuisible à la bonne santé du système d'information.

Si le synonyme consiste à désigner la même information par des noms différents, **la polysémie consiste à désigner des informations différentes sous le même nom**. Par exemple, une propriété « référence » qui représenterait ici le code d'un produit, là celui d'un fournisseur, et ailleurs celui d'un client.

Même si dans l'absolu, la polysémie n'entraîne pas obligatoirement des problèmes fatals (tant que les polysèmes figurent dans des entités différentes), elle représente tout de même un choix bien dangereux. C'est une source potentielle de confusions. Or, en informatique, on a déjà bien assez de travail avec les problèmes inévitables ; les problèmes évitables, autant... les éviter.

Pour lutter contre la polysémie, il existe un truc aussi simple qu'imparable : utiliser systématiquement des noms de propriétés **composés**, qui se termineront par le nom de la table à laquelle elles appartiennent. En quelque sorte, le nom de la propriété est un prénom, celui de la table un nom de famille ! Ainsi, par exemple, on peut avoir comme propriétés code\_disque, titre\_disque, artiste\_disque, etc. Dans le cas que j'évoquais à l'instant, on ne risquerait plus de confondre référence\_produit, référence\_fournisseur et référence\_client.

À pratiquer systématiquement donc, dès que la base commence à prendre un peu d'ampleur et que le risque de polysémie commence à montrer le bout de son nez.



### 4.5 – La recherche de la transitivité

La transitivité est cette propriété mathématique qui dit qui si A est dans une certaine relation avec B, et B avec C, alors A est en relation avec C. Appliqué à la question de la modélisation, cela veut dire qu'il faut veiller à ne pas doubler inutilement les chemins entre les tables.

Reprenons, une fois encore, le cas de notre discothèque. La situation de départ peut être modélisée par trois entités : des artistes, des CDs et des genres. Les artistes sont en relation avec les CDs ; les CDs sont en relation avec les genres. La question est : est-il bénéfique, neutre ou nocif de mettre de surcroît les artistes en relation avec les genres ?

De deux choses l'une. Soit nous ne voulons signifier rien de plus que : « tel artiste a créé des oeuvres qui relèvent de tel et tel genre, on peut donc dire que lui-même se rattache à tel et tel genre ». Mais alors, on voit très bien que cette idée se déduit de la double relation Artistes - CDs et CDs - Genres. Rajouter une relation directe entre Artistes et Genres ne ferait, dans ce cas, qu'au mieux ajouter une information inutile, au pire introduire des discordances. Soit (et même si, en l'occurrence, on peut penser que c'est un peu stupide), on se dit qu'on veut établir par avance une série de genres pour chaque artiste afin de vérifier que ses oeuvres sont bel et bien conformes à son style. Alors – mais seulement alors &ndash il y a un sens à doubler la relation Artistes - CDs et CDs - Genres par une relation directe Artistes - Genres.

On peut formuler le principe général en disant qur dans un MCD, une relation ne doit jamais être redondante par rapport à une autre (ou une série) d'autres. En clair, il ne peut y avoir qu'un seul chemin entre une entité A et une entité B, **sauf si les différents chemins correspondent à des significations différentes**. La normalisation doit donc veiller à éliminer ces éventuelles relations parasites.

On trouvera une illustration très claire de ce principe dès le prochain chapitre.





# Petit cours de Modélisation

Introduction à Merise avec exercices et corrigés

1. Cadre général
2. Premiers problèmes...
3. Entités et relations
4. La normalisation
5. Techniques avancées

Enseigné dans le [M2 PISE](#) de l'Université Paris 7 par [Christophe DARMANGEAT](#).

Toutes les erreurs, insuffisances et plaisanteries affligeantes contenues dans ce site relèvent de la seule responsabilité de la bassesse de leur auteur (NB : attention, jeu de mots).

Si ce cours vous a instruit, ou amusé, ou mieux, les deux à la fois, n'hésitez surtout pas à me remercier en m'invitant au restaurant ou sur une barrière de corail. Ou mieux, les deux à la fois.

## 5 – Techniques avancées

### 5.1 – Relations multiples

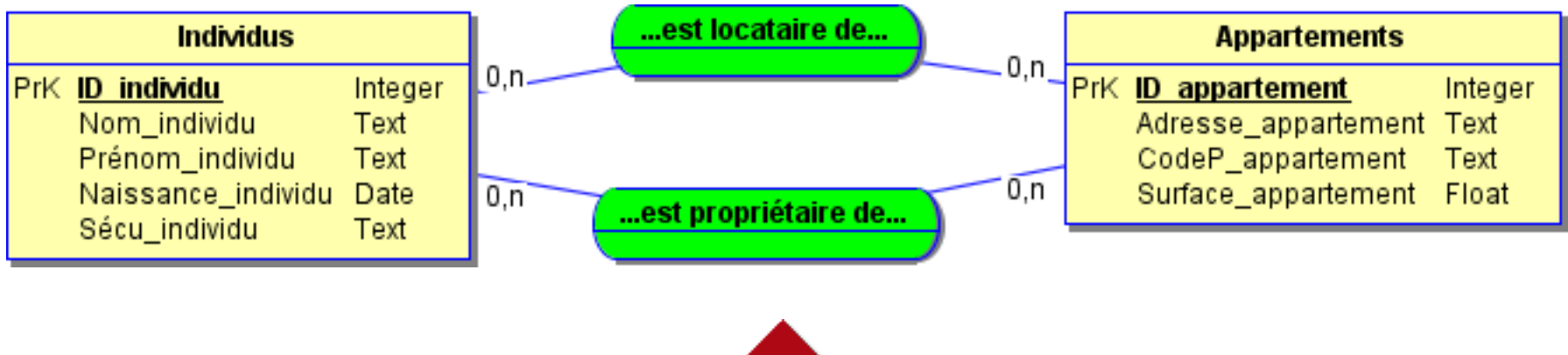
Il est tout à fait possible que deux entités soient reliées à la fois par plus d'une relation.

Une telle possibilité n'a rien d'absurde, et correspond à beaucoup de situations de la vie réelle.

Imaginons, par exemple, que nous ayons à modéliser les relations entre une population et un parc immobilier. Nous aurons deux entités : l'une qui identifiera les différents individus (et dont les propriétés seront leurs noms, prénoms, n° de sécurité sociale, etc.). L'autre entité recensera les différents logements (n° d'appartement, étage, adresse, etc.). Or, il est clair que la relation d'un individu avec un logement peut être de plusieurs natures différentes, qui sont très largement indépendantes les unes des autres :

- il peut en être propriétaire
- il peut en être locataire
- il peut en être bailleur
- il peut l'occuper à titre de résidence principale
- etc.

Face à ce problème, la solution est simple : les deux entités Individus et Logements doivent être reliées simultanément par plusieurs relations différentes. On parle alors d'**associations plurielles**. Cette technique illustre, en quelque sorte à l'envers, la question de la transitivité : il peut y avoir plusieurs relations qui mènent d'une entité à l'autre, à condition que ces relations possèdent des significations différentes.



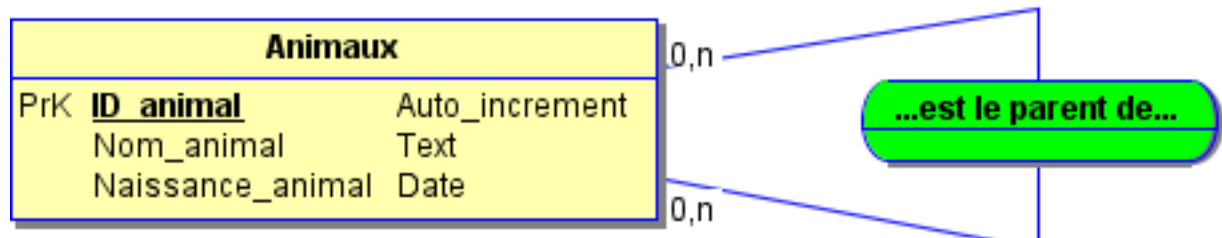
### 5.2 La réflexivité

#### 5.2.1 Principe général

Encore un mot de matheux pour désigner une chose pas si compliquée qu'elle en a l'air, à savoir qu'une relation peut concerner des éléments d'une même entité.

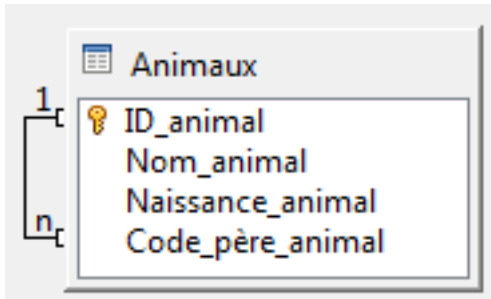
Supposons une base d'individus, par exemple les animaux d'un zoo. En vue de suivre les pedigrees de chaque bête, nous devons enregistrer qui est l'enfant de qui. Pour cela, il nous faudra relier l'entité Animaux à elle-même, par la relation « *est le parent de...* ». Cela peut paraître un peu bizarre, mais en réalité, cela ne pose aucun problème.

On pourra donc modéliser tout cela dans notre MCD de la manière suivante :

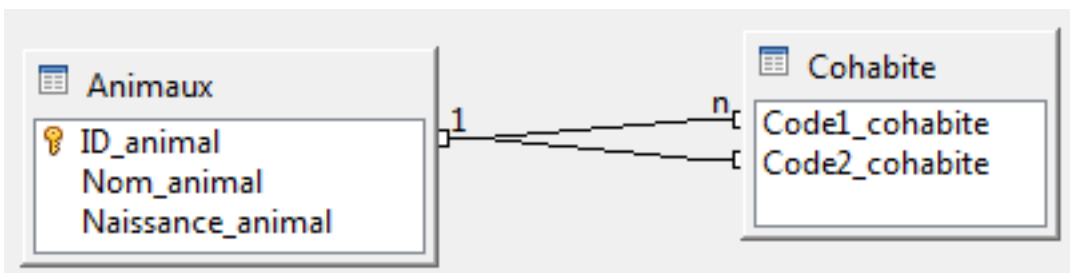


#### 5.2.2 Du MCD au MLD

Au niveau du MLD, il n'y a rien de particulier à signaler – les règles exposées plus haut s'appliquent. Si la relation réflexive est de type « un à plusieurs », avec une cardinalité minimale de 1, cette relation prendra la forme d'une propriété supplémentaire dans la table, destinée à contenir l'ID de l'élément concerné. Par exemple, la relation « est le père de », qui entre dans ce cas de figure (un père peut avoir plusieurs enfants, mais un enfant ne peut avoir qu'un seul père) se manifestera par un champ code\_père, dans lequel on entrera, le cas échéant, l'ID du père :



Si, en revanche, la relation réflexive est de type « plusieurs à plusieurs », elle donnera lieu à une nouvelle table. Celle-ci comportera deux champs, chacun des deux contenant l'ID d'un des deux animaux susceptibles de cohabiter. Et, toujours aussi logiquement, cette table de jointure sera reliée au champ ID\_animal de la table Animaux par un double lien.



**NB 1 :** On peut très bien imaginer une situation où une relation serait à la fois réflexive et plurielle ! En plus de savoir qui est le parent de qui, on pourrait ainsi noter qui peut être mis dans la même cage que qui (certains individus ne peuvent pas se blairer, et se battent dès qu'ils en ont l'occasion). Dans ce cas, pas de problème, l'entité Animaux sera greffée de deux relations réflexives différentes (« est le parent de » et « peut cohabiter avec »).

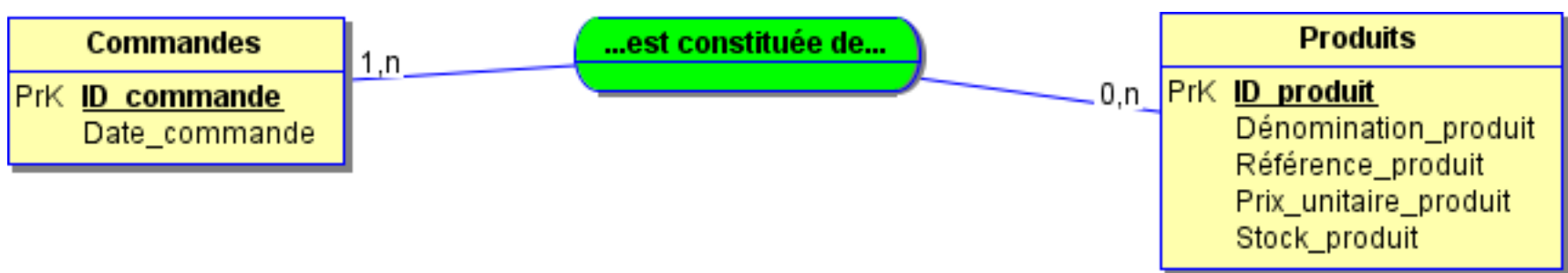
**NB 2 :** Il serait en revanche tout à fait mal venu de créer deux relations, l'une « est le parent de » et l'autre « est l'enfant de ». Ce serait en quelque sorte un pléonasme, ces deux relations n'en faisant en réalité qu'une seule.

### 5.3 Relations avec attributs

#### 5.3.1 – Principe général

Prenons le cas tout bête d'une entreprise qui gère les commandes de ses clients.

Dans une première entité figurent les commandes, avec leur date, leur référence, etc. Pour savoir de quoi est composée chaque commande, on se tourne naturellement du côté d'une entité qui regroupe l'ensemble des produits vendus par notre entreprise. Au premier abord, il semble que nous soyons face à une situation déjà rencontrée maintes fois :



Sauf qu'il y a un petit souci : je n'ai fait figurer nulle part la quantité dans laquelle chaque produit entre dans chaque commande. Par exemple, la commande n°456 peut porter sur 45 rideaux de douche à fleurs roses et 12 gants de toilette en crin, la commande n°508 sur 23 gants de toilette en crin et 3 savons de Marseille, etc. Or ces nombres (45, 12 et 23) doivent obligatoirement être stockés quelque part, faute de quoi mon système de gestion de factures ressemblera à une passoire.

Première possibilité : je je l'intègre à la table des commandes. Hum... Comment faire ? En créant un seul attribut ? Mais cela veut dire que pour une commande donnée, on ne pourra renseigner qu'un seul nombre. Or, dans une commande, on peut commander plusieurs produits çà la fois (chacun dans une quantité différente). Donc, ça ne va pas.

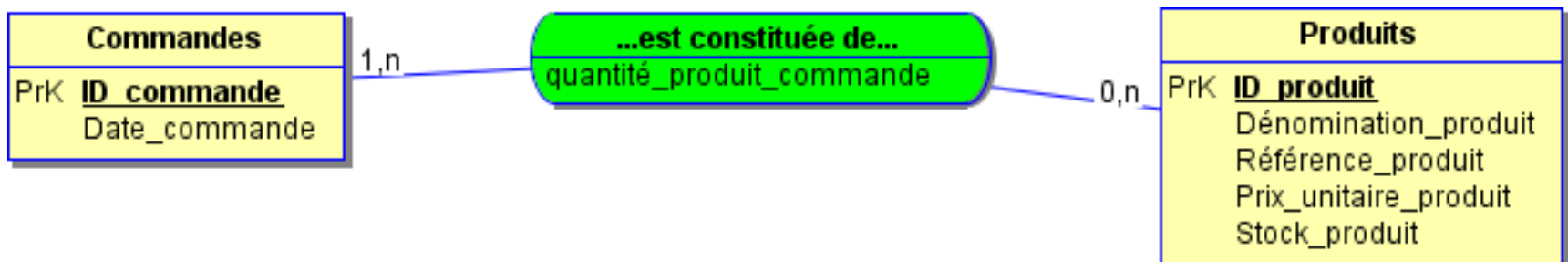
Deuxième possibilité : j'intègre les quantités à la table des produits. Mais là, c'est le même problème : Si je crée un nouvel attribut (et un seul), cela veut dire qu'à chaque produit correspond une quantité et une seule. Sauf qu'évidemment, il se peut très bien que le nombre de Kiki en peluche commandés par Bidule soit de 7 546 tandis que Machin, lui, en a acheté 12. Donc, problème.

Je n'ose même pas imaginer créer plusieurs attributs (quantité 1, quantité 2, quantité 3, etc.) : que ce soit dans une table ou dans un autre, ce serait un flagrant délit de propriété à valeurs multiples, dont on a vu précédemment qu'il s'agissait d'une faute irrémédiable.

Nous voilà donc bien avancés... En fait, toute la difficulté vient du fait que **le nombre de produits de chaque sorte qui ont été commandés est une caractéristique liée à chaque commande**. La solution s'impose d'elle-même : cette information doit donc figurer au cœur même de la relation qui unit les commandes aux produits. À chaque fois qu'une nouvelle commande porte sur un nouveau produit, on doit se demander : « *en combien d'exemplaires ?* ».

L'information (la propriété) « Quantité commandée » n'est donc pas un attribut d'une des deux entités Commandes et Produits, mais comme **un attribut** de la relation qui les unit.

Ce que nous représenterons de la manière suivante :



**Remarque vraiment intelligente :** une relation ne peut posséder légitimement un attribut que si elle est de type « plusieurs à plusieurs ». **Démonstration par l'absurde :**

- imaginons qu'il ne puisse y avoir qu'un seul produit par commande. Dans ce cas, la quantité commandée serait unique pour chaque commande, et pourrait donc être un attribut de l'entité Commande.
- inversement, si chaque produit ne pouvait être commandé qu'une fois et une seule, on pourrait ajouter la quantité comme attribut de l'entité Produits

Pour conclure, ajoutons que rien n'oblige une relation à posséder un seul attribut. Il est tout à fait possible qu'existent des relations avec deux, trois, quatre, etc. attributs.

#### 5.3.2 – Passage au MLD

Aucune difficulté : une relation avec attributs étant nécessairement une relation « plusieurs à plusieurs », elle donnera naissance, lors du passage au MLD, à une nouvelle table (de jointure). Tout simplement, **les attributs de la relation deviendront des propriétés supplémentaires de la table de jointure** (en plus des codes correspondant aux clés primaires des tables concernées).

## 5.4 – Relations de dimension supérieure à 2

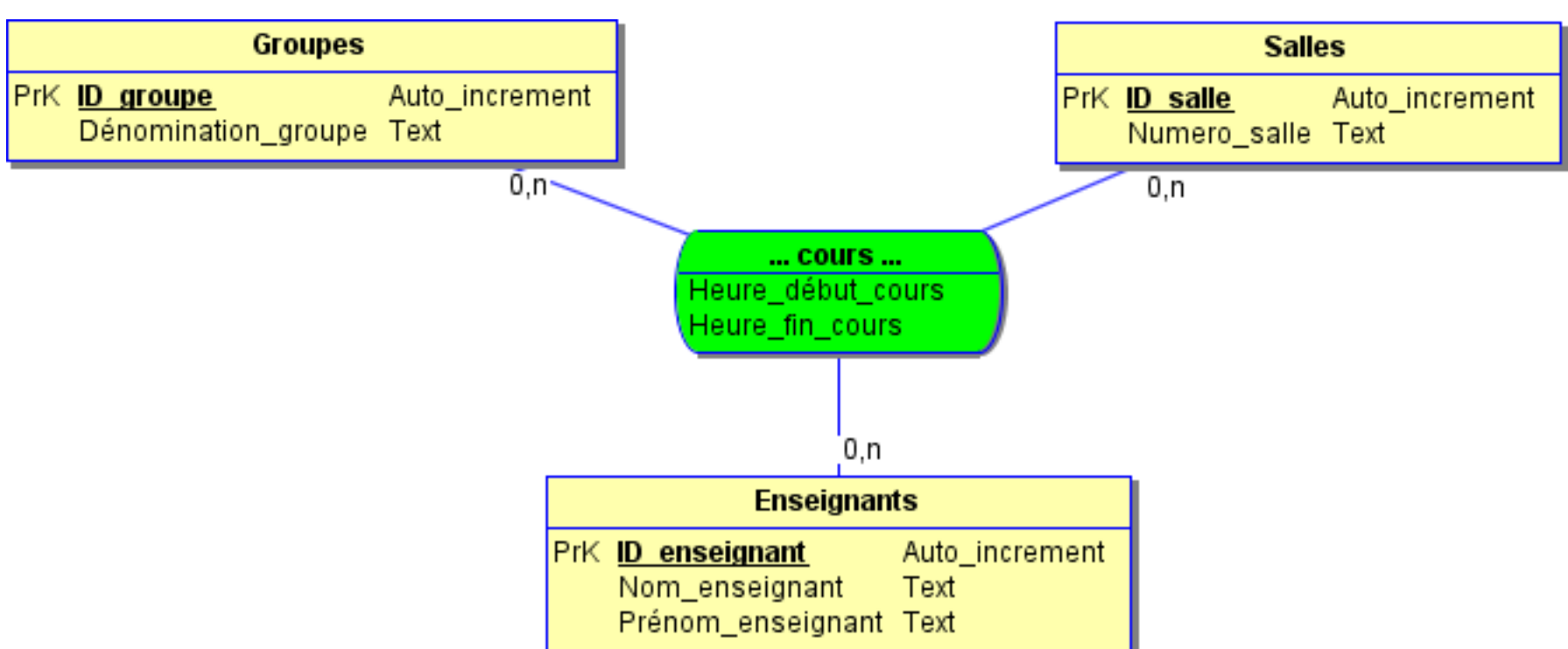
Aïe aïe aïe, encore du vocabule matheux. Mais non, c'est rien, ça pique à peine au début et après on ne sent plus rien.

#### 5.4.1 – Principe général

Partons simplement du fait que jusqu'à présent, les relations que nous avons modélisées ne mettaient toujours aux prises que des entités deux à deux : les CD et les genres musicaux, les commandes et les produits, etc. Il peut cependant arriver que la relation doive s'établir directement entre plus de deux entités (d'où le titre).

Prenons un système d'information qui gère le travail quotidien dans une université (bon courage). Chaque jour, des groupes d'étudiants ont cours dans certaines salles avec certains professeurs. Mais on suppose que tout dans cette histoire est susceptible de varier : dans une journée donnée, le même groupe peut avoir cours avec plusieurs profs dans plusieurs salles différentes, les profs peuvent donner plusieurs cours ou aucun, etc.

La manière la plus directe de modéliser une telle situation sera d'écrire que :



On est donc face à une relation impliquant non plus deux, mais trois entités, ce qui finalement, ne pose pas de problèmes particuliers. Au passage, on remarque qu'une relation de dimension supérieure à 2 peut fort bien comporter des attributs, comme toute autre relation (Ici, Heure\_début\_cours et Heure\_fin\_cours).

#### 5.4.2 – Du MCD au MLD

On vérifie facilement qu'une relation de dimension supérieure à deux est forcément de type « plusieurs à plusieurs » (faudrait-il dire : « plusieurs à plusieurs à plusieurs » ? – en voilà une question qu'elle est bonne). Une telle relation devient donc systématiquement une table de jointure, comportant autant de champs que d'entités à relier.

## 6. – Esquisse d'une méthode générale

Si l'on tente de résumer l'ensemble des questions à se poser pour réussir une modélisation, on obtient quelque chose qui pourrait ressembler à ceci :

1. Identifier l'ensemble des entités. Décomposer celles-ci autant que nécessaire, en se demandant si, pour chacune d'elles, il existe des propriétés dont les valeurs seront redondantes. Si oui, 99 fois sur 100, ces propriétés doivent être « externalisées » vers une nouvelle entité.
2. Traquer et éliminer les valeurs multiples et les synonymes
3. Quelles sont les relations ? Pour chaque relation établie entre les entités A et B, se demander : « un élément de A entre-t-il plusieurs fois en relation avec le même élément de B ? » Si non, c'est une relation simple (dont on examine ensuite les cardinalités). Si oui, quelles sont les informations qui différencient ces différentes associations. Si ces informations ne sont pas particulièrement redondantes, elles sont des attributs de la relation. Si elles sont redondantes, elles doivent être issues d'une entité, et la relation est donc de dimension > 2. Dans le cas où cette étape fait surgir une nouvelle entité, reprendre l'étape précédente.
4. Vérifier que toute relation multiple, directe ou indirecte, d'une entité à une autre, se justifie par le fait que chacune d'elles exprime des significations différentes.
5. Traquer et éliminer les polysèmes

Allez, là, ça va carrément être l'extase :

- [exercice 3](#)
- [exercice 4](#)
- [exercice 5](#)
- [exercice 6](#)
- [exercice 7](#)
- [exercice 8](#)