# 6. La normalisation des données

## Les formes normales: grammaire de la modélisation des données

Nous l'avons déjà dit: les formes normales de Codd – Date constituent les règles de grammaire du monde de la normalisation des données. Nous avons aussi dit: attention, de même qu'un texte peut être totalement correct du point de vue grammatical et ne contenir que des âneries, il ne suffit pas à un modèle des données d'être entièrement normalisé pour être correct du point de vue des besoins de l'application ou de l'entreprise. En réalité, les formes normales ne nous aident que très peu à concevoir la base de données, mais servent plutôt de garde-fou pour ne pas mettre en place des structures qui poseront des problèmes à l'exploitation.

Émises par E.F. Codd en 1970, les règles de normalisation révolutionnèrent la manière de stocker les données sur un ordinateur et constituèrent la base pour l'avènement des SGBD relationnels. À l'époque, ces principes étaient pourtant si novateurs que beaucoup pensaient qu'il serait impossible de les appliquer sans restriction dans la pratique. Les problèmes liés au manque d'espace disque, aux performances et à la programmation étaient simplement insurmontables: "le relationnel est trop lent" disait-on. La normalisation a en effet pour conséquence de multiplier le nombre de tables dans lesquelles les données sont (correctement) réparties. Pour réaliser les applications, il faut alors réunir des valeurs provenant de beaucoup de tables, d'où (pensait-on alors) une plus grande complexité de programmation et une moindre efficacité d'exécution. Aujourd'hui le modèle relationnel est si répandu et les outils si performants que plus personne ne songe à faire différemment.

À l'époque on parlait de "normaliser" les données. Partant de structures de données existantes qui n'obéissaient à aucune loi, on les soumettait au processus de mise en première forme normale, puis en deuxième forme normale, puis en troisième. L'auteur se souvient même d'avoir entendu un directeur de service informatique affirmer que: "nous normalisons nos données, mais n'allons pas plus loin que la deuxième forme normale ". Il n'avait manifestement rien compris, si ce n'est la nature progressive de la théorie: une structure de données n'est en effet dite en $2^{ème}$ forme normale que si elle est aussi en $1^{ère}$ et en $3^{ème}$ forme normale que si elle est également déjà en $1^{ère}$ et $2^{ème}$. Quelques fois, placé devant des situations de performances inacceptables, on se résignait finalement à "dénormaliser" à nouveau, ce qui revenait en pratique à stocker une même information en plusieurs endroits. Malheureusement il arrive aujourd'hui encore, mais rarement, qu'il faille recourir à de tels procédés. Normaliser revient en effet, mais pas seulement, à n'enregistrer la même information qu'en un seul et unique endroit dans la base de données, nous y reviendrons.

Le lecteur l'aura compris: il existe trois formes normales: la première, la seconde et la troisième. Dans le jargon on les appelle quelques fois FN1, FN2 et FN3, ou NF1, 2 3 en anglais. Comme nous le verrons, il existe même d'autres formes normales, mais leur importance est marginale en pratique.

Aujourd'hui nous ne sommes généralement plus dans une situation où il s'agit de purifier des structures aberrantes du passé. **On ne "normalise" donc plus, mais on conçoit au contraire dès le départ des structures propres, conformes aux lois de Codd – Date**. Ces lois servent de garde-fou et l'on vérifie en permanence que les définitions des types d'objets leur sont conformes.

Du fait que la $1^{ère}$ forme normale a été abolie formellement il y a quelques années (mais reste appliquée dans la majorité des cas pour des raisons de simplicité), ainsi que de ce qui a été dit au paragraphe précédent, les préoccupations liées au respect des formes normales n'ont certes pas disparu, mais sont plus ou moins reléguées à l'arrière-plan. Créer des structures normalisées est devenu un réflexe.

Terminons-en avec ce prélude en réglant nos comptes avec le monde de la formation. La thématique des formes normales constitue évidemment du pain bénit pour des maîtres d'école de tout rang. Voilà un sujet qui se prête admirablement à des livres, cours, exercices et examens, un moyen de faire bosser les étudiants et de trier le bon grain de l'ivraie. Peu importe que ni formateurs ni formés aient la moindre connaissance des domaines sur lesquels portent les exemples utilisés, certaines solutions sont jugées correctes et d'autres incorrectes, même si elles n'ont rien à voir avec la réalité. Nous avons sou-

vent eu l'occasion d'admirer des solutions délirantes conçues par des professeurs, véritables ayatollahs de la normalisation, dont il était bien évident qu'ils n'avaient jamais franchi le seuil d'une entreprise ou d'une organisation réelle.

Tout ceci pour rappeler une fois encore que la vision des concepteurs d'une base de données doit porter sur l'application et les besoins de l'entreprise et des utilisateurs. Évidemment dans le respect des formes normales. Mais la modélisation des données dans la réalité ne doit pas et ne peut pas se réduire à un exercice de normalisation.

## Buts de la normalisation

On exige aujourd'hui que les structures de données dans une base de données soient normalisées. Ceci pour les raisons suivantes:

- Une structure normalisée se laisse idéalement implémenter au moyen d'un SGBD relationnel et exploiter au moyen des outils existants, notamment le langage SQL.
- Dans une structure normalisée, chaque élément d'information n'est enregistré qu'une seule fois, en un seul endroit de la base. Si la valeur de cette information change, elle ne doit donc être modifiée qu'en un seul endroit.
- Les structures normalisées sont plus simples à comprendre et le développement de programmes est plus efficace avec les outils actuels.
- Une structure normalisée peut être plus facilement modifiée et étendue. Elle ne comporte pas de limitations de principe.
- Une base de données à structure normalisée ne contient pas de contradictions (du fait que chaque information n'est enregistrée qu'en un seul endroit).

Une base de données construite sur une structure non normalisée présente les problèmes suivants:

- Elle peut contenir des redondances, des anomalies (dépendances ne correspondant pas à la réalité) et des contradictions.
- Elle peut contenir des limitations qui peuvent forcer des modifications de structure lorsque le volume de données augmente
- La réalisation des programmes est plus complexe.

## Dépendance fonctionnelle

Pour comprendre les formes normales, il faut comprendre la notion de **dépendance fonctionnelle** entre attributs.

**Un attribut B est dit fonctionnellement dépendant d'un autre attribut A si la valeur de B est déterminée par la valeur de A**.

Autrement dit: si la valeur de A est connue, la valeur de B l'est également (il suffit de consulter la ligne correspondante de la table).

Exemples:

Personne (#-personne, nom, prénom, sexe, date-naissance, rue, ville)

Tous les attributs *nom, prénom, …, ville* sont fonctionnellement dépendants de *#-personne*: dès que *#-personne* est connu, la valeur des autres attributs est totalement déterminée, car chaque personne ne possède qu'un nom, prénom, etc. Dans la table *Personne*, chaque personne est identifiée par *#-personne* qui détermine la ligne de la table où l'on retrouve son nom, etc.

Par contre, le fait de connaître la valeur d'un autre attribut, *nom*, par exemple, ne détermine pas la valeur des autres, puisque plusieurs personnes peuvent porter le même nom, plusieurs lignes dans la table peuvent contenir la même valeur de nom. De même, connaître *prénom* ne détermine pas *sexe*,

puisque certains prénoms sont attribués aux deux genres (Dominique, George, ou encore Carol dans le monde anglo-saxon, etc.).

**La clé primaire (ou un candidat à la clé primaire) détermine la valeur de tous les autres attributs d'un type d'objet. Tous les autres attributs doivent être fonctionnellement dépendants de la clé primaire.**

## Résumé des formes normales de Codd – Date

Les trois principales formes normales vont être discutées en détail et illustrées avec des exemples dans la suite de ce chapitre. Ici elles sont simplement résumées. Ce résumé ne permettra probablement pas à une personne confrontée à ce sujet pour la première fois d'en comprendre le sens et la portée. Que ces personnes se reportent aux présentations détaillées qui suivent.

Toutes les définitions de types d'objets doivent obéir aux règles suivantes:

- Les types d'objets sont représentés dans la base de données sous forme de **tables plates** (à 2 dimensions). Les colonnes de la table correspondent aux attributs, les lignes aux objets enregistrés.
- Chaque table possède une clé primaire formée d'un ou de plusieurs attributs. La valeur de **la clé primaire est unique** dans la table et détermine l'objet.
- Chaque attribut est **atomique** (sans structure interne) et chaque attribut n'apparaît qu'une seule fois dans la définition du type d'objet (pas de **tables** d'attributs). C'est la 1ère forme normale.
- Si la clé primaire d'une table est formée de plusieurs attributs, la valeur de chaque attribut de cette table dépend fonctionnellement de **l'entièreté de la clé** et non pas seulement d'une **partie** des attributs constituants la clé. C'est la 2ème forme normale.
- Chaque attribut figurant dans la définition d'un type d'objet ne dépend fonctionnellement que de la **clé primaire** et non pas d'un **autre** attribut faisant partie de la définition de ce type d'objet. C'est la 3ème forme normale.

En allusion au serment prononcé par des témoins au tribunal dans les films américains: "the truth, the whole truth and nothing but the truth, so help me God" (la vérité, toute la vérité et rien que la vérité, que Dieu me vienne en aide), on résume quelques fois les formes normales de la façon suivante:

"The key, the whole key and nothing but the key, so help me Codd" (la clé, toute la clé et rien que la clé, que Codd me vienne en aide).

## 6.1. Première forme normale de Codd – Date

**Dans une définition de type d'objet, chaque attribut doit être atomique (sans structure interne) et chaque attribut doit être différent: les tables d'attributs ne sont pas admises.**

**Commentaires:** (destinés aux personnes qui connaissant déjà le sujet)

Lorsque Codd et Date ont émis leur théorie des bases de données relationnelles en 1970, cette première forme normale servait avant tout de guide pour créer des structures de données simples, par opposition aux structures complexes souvent mises en œuvre à l'époque. Elle introduisait le concept de table "plate" (à deux dimensions) dans laquelle chaque colonne représente un attribut et chaque ligne un objet. Du point de vue formel, cette première forme normale n'a rien à voir avec les suivantes. Après des années de tergiversations, les deux auteurs ont donc été forcés de faire machine arrière, en déclarant, ce qui leur a permis de sauver la face, que "toutes les définitions d'objets sont automatiquement en 1ère forme normale". L'influence de l'orientation objet et d'autres avancées théoriques ont ensuite encore réduit l'importance de la table plate. Les nouvelles versions de SQL (langage d'interrogation et de manipulation des données dans les SGBD relationnels) permettent aujourd'hui de manipuler des structures non plates, donc des attributs à structure interne et des tables d'attributs.

Néanmoins, du point de vue pratique et pour des raisons de simplicité, il est fortement recommandé de s'en tenir à la 1ère forme normale et au concept des tables plates. De toute manière, les débutants ne feront pas autrement. **Fin des commentaires.**

**Exemple 1.** La définition suivante n'est pas en 1ère forme normale:

Personne (#-personne, nom-prénom, date-naissance, sexe, adresse)

Elle n'est pas en 1ère forme normale parce que les attributs *nom-prénom* et *adresse* ne sont pas atomiques. Parlons d'abord du second: une adresse se compose dans la pratique de plusieurs éléments tels que *rue, code postal, localité, pays*. On peut bien imaginer des applications simples où elle se réduirait à une localité (mieux vaut alors changer le nom de l'attribut pour être précis), mais toute gestion d'adresse sérieuse nous oblige à décomposer cet attribut.

De même pour *nom-prénom*. Procéder de la sorte fera que certaines lignes de la table contiendront le prénom suivi du nom et d'autres le nom suivi du prénom, ne permettant pas de trier correctement les listes de personnes. Sans oublier que certains prénoms sont aussi des noms de famille, ce qui ajoutera encore à la confusion.

Pour être précis et éviter les ennuis nous recommandons de déstructurer tous les attributs dans la mesure du possible.

Voici donc une meilleure solution:

Personne (#-personne, nom, prénom, date-naissance, sexe, rue, no, code-localité, nom-localité)

(Nous renvoyons à plus tard la discussion de savoir si l'attribut *nom-localité* est nécessaire du fait qu'il est probablement déterminé par code-localité).

[Les outils modernes nous permettent pourtant aussi de gérer les attributs non-atomiques. Dans ce cas, nous formulerions les choses de la façon suivante:

Personne (#-personne, noms (nom, prénom), … , adresse (rue, no, code, localité))

Comme nous l'avons déjà remarqué, nous conseillons vivement de s'en tenir à la version simple qui figure plus haut.]

**Exemple 2:** La définition suivante n'est pas en première forme normale:

Personne (#-personne, nom, prénom, sexe, date-naissance, langue 1, langue 2, langue 3)

Cette définition n'est pas en 1ère forme normale parce qu'elle contient une table, la table des langues parlées par ces personnes.

Procéder de la sorte amène plusieurs inconvénients ou problèmes:

- Une personne ne peut pas parler plus de 3 langues
- Si une personne parle moins de 3 langues, certaines valeurs de la table resteront vides, d'où place gaspillée
- Si on recherche les personnes parlant une certaine langue, on devra consulter trois colonnes de la table
- Pas possible de qualifier (à moins de compliquer encore) le degré de connaissance par la personne de la langue en question. Les tables dans les définitions de types d'objet souffrent souvent de ce défaut: les éléments de la table sont eux-mêmes des objets qui nécessitent d'être décrits au moyen d'attributs.

Une meilleure solution est de créer deux tables:

Personne (#-personne, nom, prénom, sexe, date-naissance)

Langue parlée (*#-personne*, langue, degré de connaissance)

Dans langue parlée, *#-personne* est à la fois clé étrangère faisant le lien avec la clé primaire de *personne* et partie de la clé primaire. Chaque langue parlée par chaque personne résulte en une ligne dans la table *Langue parlée*. Une telle solution peut à première vue paraître plus compliquée que celle violant la 1ère forme normale, mais a) elle est propre et exempte de limites, b) elle est plus facile à exploiter avec les outils modernes. Cette solution sera encore améliorée par la suite.

Voici le résultat:

| **Personne** | | | | |
|---|---|---|---|---|
| **#-personne** | **Nom** | **Prénom** | **Sexe** | **Date-naissance** |
| 1001 | Durand | Anne | F | 3-10-1953 |
| 1002 | Renggli | Gilbert | M | 7-5-1940 |
| 1003 | Von Erlach | Georg | M | 9-3-1946 |

| **Langue parlée** | | |
|---|---|---|
| **#-personne** | **Langue** | **Degré de connaissance** |
| 1001 | Français | Langue maternelle |
| 1001 | Allemand | Oral |
| 1002 | Français | Langue maternelle |
| 1002 | Anglais | Excellent |
| 1003 | Allemand | Langue maternelle |
| 1003 | Français | Excellent |
| 1004 | Anglais | Excellent |

Par la suite, nous allons encore formaliser (codifier) les valeurs de *langue* et *degré de connaissance*.

Les violations de la 1ère forme normale sont généralement faciles à reconnaître. Mais il existe pourtant de nombreux cas limites. Qu'en est-il des situations suivantes?

Article (#-article, désignation, …, fournisseur, 2ème fournisseur, …)

Article (#-article, désignation, …, prix, dernier prix, …)

Faut-il dans tous les cas créer une table des fournisseurs potentiels pour un article, et une table contenant l'historique des prix? Seule une analyse détaillée des besoins pourra nous le dire.

Mais réfléchir à la question de savoir si une table des fournisseurs potentiels pour un article (on parle aussi de sources d'approvisionnement) est nécessaire ou non nous fait découvrir toute une problématique que nous aurions peut être omise: faut-il gérer les sources d'approvisionnement en détail ou non, et si oui, dans quelle mesure? Voilà une problématique à analyser en détail et qui nous fait découvrir une réalité: les discussions concernant la définition des types d'objets et la normalisation nous amènent souvent à découvrir des aspects inattendus des solutions à l'étude.

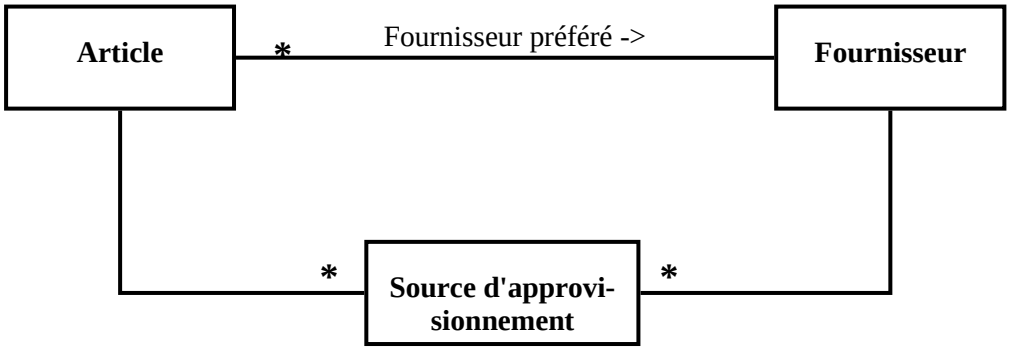
Voilà ce que pourrait donner une solution complète:

Article (#-article, désignation, …) simple catalogue des articles

Fournisseur (#-fournisseur, raison sociale, …) simple liste d'adresses de fournisseurs

Source d'approvisionnement (*#-article, #-fournisseur*, rang, prix, délai, …)

*Rang* spécifierait ici quel est le fournisseur préféré pour un article. Une autre solution consisterait à inclure *#-fournisseur préféré* dans la table *Articles*. Graphiquement, la solution se présenterait ainsi.




## Quantités dérivées redondantes

Peut-on inclure dans les définitions des attributs qui contiennent des valeurs se laissant calculer à partir d'autres données? Le modèle ne l'interdit pas formellement, même si une interprétation rigoriste nous ferait répondre à cette question par la négative.

**Exemples: solde d'un compte, niveau de stock**

Compte (#-compte, libellé, solde)

Transaction (#-transaction, *#-compte*, date, code crédit/débit, montant)

*Solde* dans *compte* se laisse à tout moment recalculer à partir des transactions.

Article (#-article, désignation, …, niveau de stock, …)

Mouvement de stock (#-mouvement, *#-article*, date, quantité, code entrée/sortie)

Le niveau de stock se laisse à tout moment recalculer à partir des mouvements.

De telles solutions compliquent la programmation parce qu'il faut lors de chaque transaction recalculer le solde. Il en est de même lors de l'extourne d'une transaction ou de sa modification (eh oui, de pareilles choses existent!). Tôt ou tard on aura des situations dans lesquelles le total ne sera plus correct. Et

que faire lorsqu'on archive de vieilles transactions pour ne pas garder trop de données dans la base? On pourrait donc adopter une attitude très négative à l'égard de telles solutions.

D'autre part il est impensable de recalculer le solde d'un compte ou le niveau de stock d'un article (à partir de milliers de mouvements?) chaque fois que quelqu'un désire consulter une telle valeur.

Heureusement, certains SGBD (mais pas tous) nous permettent de spécifier des règles d'intégrité (on les appelle dans ce cas de procédures enregistrées) qui calculent automatiquement ces quantités dérivées et assurent donc l'intégrité de leur valeur. En l'absence de telles fonctionnalités, une analyse précise et une réalisation sans faille seront indispensables.

## 6.2. Deuxième forme normale de Codd – Date

La 2ème forme normale interdit toute dépendance fonctionnelle d'une **partie seulement** de la clé primaire. Chaque attribut doit dépendre de la clé primaire **entière**.

Elle ne s'applique donc qu'aux définitions de types d'objets dont la clé primaire est composée de plusieurs attributs.

Comme exemple de violation de la 2ème forme normale reprenons l'exemple des langues parlées figurant dans la discussion de la 1ère forme normale et ajoutons-y deux attributs.

Langue parlée (*#-personne*, langue, degré de connaissance, nom, prénom)

*Nom* et *prénom* violent la 2ème forme normale parce qu'ils ne dépendent que de la 1ère partie de la clé primaire, *#-personne*, mais pas de la 2ème, *langue*.

Voici une autre merveille, courante par le passé (il s'agit de rapports d'activités):

Rapport (*#-employé*, *#-projet*, année-mois, nom-employé, nombre d'heures, nom-projet)

Le nom de l'employé ne dépend pas du projet sur lequel il travaille et le nom du projet ne dépend pas de l'employé. Ces deux attributs ne dépendent que d'une partie de la clé primaire et violent donc la 2ème forme normale.

Solution:

Employé (#-employé, nom, prénom, …)

Projet (#-projet, nom-projet, …)

Rapport (*#-employé*, *#-projet*, année-mois, nombre d'heures)

Un esprit critique pourrait insinuer que, puisque la 2ème forme normale ne s'applique qu'aux définitions comprenant des clés primaires composées, il suffit de ne pas spécifier de types d'objets avec une clé composée, ce qu'il est toujours possible de faire (il suffirait d'ajouter une clé neutre, par exemple *#-rapport* dans l'exemple précédent). Ainsi:

Rapport (#-rapport, *#-employé*, *#-projet*, année-mois, nom-employé, nombre d'heures, nom-projet)

Malheureusement la chose n'est pas aussi simple, car on ne fait ainsi que transformer une violation de 2ème forme normale en violation de 3ème forme normale (voir section suivante)! En réalité, pour être précis, la 2ème forme normale interdit également les dépendances partielles de tout **candidat** à la clé primaire (forme normale de Boyce-Codd, voir plus loin): le regroupement de *#-employé*, *#-projet*, *année-mois* est un candidat valable à la clé primaire et tous les autres attributs doivent donc dépendre de l'entièreté de ce regroupement.

## 6.3. Troisième forme normale de Codd – Date

La 3ème forme normale de Codd – Date interdit qu'un attribut dépende **d'un autre attribut que la clé primaire**. **Chaque attribut doit dépendre exclusivement de la clé primaire** (ou d'un candidat à la clé primaire).

Exemple: la définition suivante viole la 3ème forme normale:

Employé (#-employé, nom, prénom, sexe, date-naissance, #-département, nom-département)

L'attribut *#-département* indique dans quel département travaille l'employé et il est ici à sa place. Mais le nom de ce département ne dépend clairement pas de l'employé. Il dépend de *#-département* et non de *#-employé*. Il viole donc la 3ème forme normale.

Voici ce qui pourrait arriver si l'on gardait cette définition inchangée:

| **Employé** | | | | | | |
|---|---|---|---|---|---|---|
| **#-employé** | **Nom** | **Prénom** | **Sexe** | **Date-naissance** | **#-dép.** | **Nom-dép.** |
| 1001 | Durand | Anne | F | 3-10-1953 | 1 | Finances |
| 1002 | Renggli | Gilbert | M | 7-5-1980 | 3 | Production |
| 1003 | Von Erlach | Georg | M | 9-3-1946 | 5 | Administration |
| 1004 | Perrot | Rosa | F | 8-8-1960 | 2 | Laboratoire |
| 1005 | Meier | Adda | F | 5-12-1970 | 3 | Producton (?) |
| 1006 | Rouge | René | M | 3-9-1979 | 4 | Ventes |
| 1007 | Lenoir | Marie | F | 2-2-1990 | 4 | Laboratoire (?) |

Outre qu'il n'est pas logique de faire figurer le nom du département dans la définition de l'employé, on voit que des violations de la 3ème forme normale (et de la 2ème):

a) Conduisent à des répétitions d'informations. Le nom du département est répété pour chaque employé au lieu d'être défini une seule fois.

b) Conduisent à des contradictions et erreurs. On en voit plusieurs exemples dans le tableau ci-dessus.

La solution correcte est évidemment:

Employé (#-employé, nom, prénom, sexe, date-naissance, *#-département*)

Département (#-département, nom-département)

## 6.4. Forme normale de Boyce – Codd (BCNF)

Par rapport aux formes normales 1 – 3 qu'elle contient, la forme normale de Boyce – Codd ne fait que répéter ce que nous avons déjà mentionné lors de la discussion de la 2ème forme normale.

**Tout attribut ne doit dépendre fonctionnellement que de la clé primaire ou d'un candidat à la clé primaire**.

La seule exigence pour être candidat à la clé primaire est qu'un attribut ou groupe d'attributs possède **par définition** une valeur unique pour l'ensemble des objets décrits par un type d'objet. Plusieurs attributs ou groupes d'attributs peuvent remplir cette définition et donc être des candidats pour être sélectionnés comme clé primaire.

Nous avons déjà donné un exemple lors de la discussion de la 2ème forme normale. Autre exemple:

Article (#-article, code-article, désignation, prix, …)

*#-article* (simple numérotation de chaque article) et *code-article* (appellation "parlante" de l'article, utilisée dans l'entreprise pour désigner l'article) sont tous deux uniques. Chaque article possède en effet un #-article et un code-article différents. Les deux attributs sont donc candidats à la clé primaire, même si le second n'est pas un bon choix parce que susceptible de changer un jour. Il faudra donc bien vérifier que chaque autre attribut de la définition ne dépende que de ces deux candidats.

## 6.5. 4ème forme normale

Ces deux formes normales correspondent à des situations rarissimes. Le seul exemple réaliste que nous ayons rencontré concerne des bases de données multidimensionnelles pour les systèmes d'aide à la décision.

**La 4ème forme normale interdit les dépendances multiples disjointes.**

Ainsi la définition:

Rapport (*#-employé*, *no-projet*, an-mois, heures)

contient trois dépendances multiples: employé – projet, employé – an-mois et projet – an-mois).

Elle est en 4ème forme normale parce que le nombre d'heures dépend réellement de l'employé **et** du projet **et** de la période (an-mois).

Par contre la définition:

Cours (cours, professeur, manuel)

n'est pas en 4ème forme normale si tous les professeurs utilisent le même manuel pour un cours donné.

Il faut alors "normaliser":

| | |
|---|---|
| Cours-prof (cours, professeur) | Quel cours sont enseignés par quels professeurs |
| Cours-manuel (cours, manuel) | Quels manuels sont utilisés pour quel cours |

## 6.6. 5ème forme normale

**La 5ème forme normale interdit la présence de plusieurs dépendances multiples s'il est possible d'effectuer une décomposition sans perte d'information.**

Nous reprenons les mêmes exemples:

Rapport (***#-employé, no-projet,*** **an-mois**, heures)

n'est pas décomposable sans perte d'information.

Pour ce qui est des cours, profs et manuels, avec les spécifications suivantes:

a) Un cours est enseigné par plusieurs profs et chaque prof enseigne plusieurs cours
b) La liste des manuels autorisés pour chaque cours est prescrite
c) Chaque prof utilise une partie des manuels autorisés.

Alors, la définition:

Cours (#-cours, #-prof, #manuel)

n'est pas en 5ème forme normale. La solution correcte est:

| | |
|---|---|
| Cours-prof (#-cours, #-prof) | Qui enseigne quoi |
| Manuels autorisés (#-cours, #-manuel) | Quel manuel est autorisé pour quel cours |
| Manuels utilisés (#-cours, #-prof, #-manuel) | Quel prof utilise quel manuel pour quel cours |

On le voit, il ne s'agit pas de situations qui arrivent quotidiennement!

## 6.7. Grand exercice!

Nous l'avons déjà dit: le danger avec la normalisation est de servir d'oreiller de paresse pour ne pas vraiment réfléchir et oublier les besoins réels de l'application. Voici donc un exercice qui démontre bien qu'on ne peut jamais faire l'économie d'une réflexion approfondie. Intentionnellement, nous poussons l'analyse à l'extrême.

Nous reprenons ici l'exemple des rapports de travail décrivant le nombre d'heures consacrées par nos collaborateurs à des projets effectués pour des clients.

Chaque collaborateur peut travailler à plusieurs projets et chaque projet peut être traité par plusieurs collaborateurs. Chaque projet n'appartient qu'à un seul client, mais il peut exister plusieurs projets pour un même client. Les rapports de travail se font mensuellement.

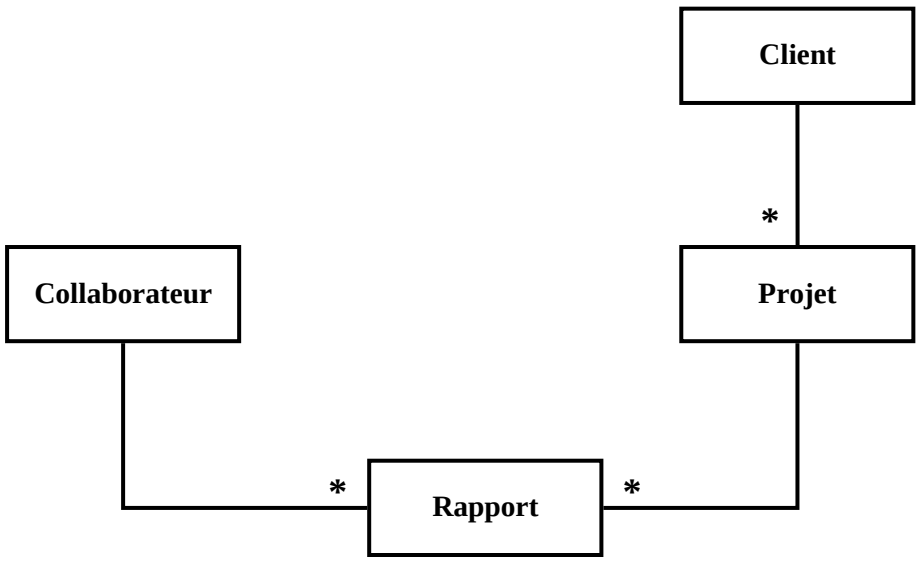
Voici la liste des définitions:

Collaborateur (#-collaborateur, nom, prénom, sexe, date-naissance, …)

Client (#-client, raison sociale, adresse, …)

Projet (#-projet, nom-projet, *#-client*, date-début, …)

Rapport (#-rapport, *#-employé*, *#-projet*, an-mois, heures)




Il s'agit à présent de facturer au client les travaux effectués. Mais à quel prix?

Question: Trouver où figurera l'attribut *prix facturé*? Que le lecteur réfléchisse un moment à cette question avant de lire la discussion qui suit.

La seule réponse correcte est: "ça dépend". (Réponse typique de consultant!)

L'étude des formes normales ne nous apporte malheureusement que très peu d'aide ici, sinon de rappeler que l'important est **de quoi** dépend ce prix.

**Éléments de réponse:**

Si l'on utilise dans l'entreprise une grille de tarifs applicables aux projets et aux clients, mieux vaut définir une table des tarifs, par exemple:

Tarif (#-tarif, libellé tarif, prix)

Ce *#-tarif* devra ensuite figurer comme clé étrangère dans le client, le projet, ou le rapport (voir discussion qui suit). L'avantage est de pouvoir modifier le prix en un seul endroit.

Si par contre une table des tarifs ne se justifie pas, il faudra faire directement figurer le prix à l'endroit voulu, tenant compte des arguments qui suivent.

Si le prix est convenu pour un client, quel que soit le projet ou le collaborateur, le prix devra figurer dans le client:

Client (#-client, raison sociale, adresse, …, prix-client)

Évidemment, si le prix facturé est par projet, il devra figurer dans le projet:

Projet (#-projet, nom-projet, *#-client*, date-début, …, prix-projet)

Si nous appliquons des prix différents par collaborateur, ce prix devra figurer dans le collaborateur:

Collaborateur (#-employé, nom, prénom, sexe, date-naissance, …, prix collaborateur)

Si nous avons convenu avec un client pour un projet des prix différents pour chaque collaborateur:

Prix projet – collaborateur (*#-projet, # collaborateur*, prix)

Et si, finalement, nous désirons pouvoir adapter le prix facturé quoiqu'il arrive (par exemple pour des travaux effectués la nuit ou le week-end ou parce que certaines heures ne peuvent pas être facturées ou doivent l'être à tarif réduit):

Rapport (#-rapport, *#-employé*, *#-projet*, an-mois, heures, prix facturé)

D'autres possibilités existent encore. Peut être faudra-t-il adopter non pas un, mais plusieurs des cas qui précèdent. Et nous n'avons pas abordé le problématique des prix qui changent d'une période à une autre, à savoir que tel prix sera applicable jusqu'à telle date et tel autre prix par la suite.

On le voit, seule une analyse approfondie des besoins de l'application nous apportera (peut être) une réponse valable. Et elle sera différente pour chaque entreprise.

(Il existe sur le marché des progiciels de gestion des temps de travail, notamment pour les professions libérales. De tels outils doivent évidemment laisser une grande flexibilité au niveau de la tarification).