Depuis sa première version 16 bits, jusqu'à la version 32 bits actuelle (V4), Delphi est livré avec un moteur de base de données de type Relationnel : le BDE. Les grandes fonctions de ce moteur sont accessibles via des composants Delphi, ce qui en cache la sophistication mais aussi le fonctionnement intime.

BDE: Borland Database Engine, Moteur de Base de Données Borland

La banalisation bénéfique des composants fait que, visuellement depuis la palette, un TTable occupe une place ni plus ni moins importante qu'un TLabel. Pourtant l'un est une porte d'entrée vers le BDE et l'autre n'est qu'un bout de texte affiché à l'écran.

Les SGBD-R, avec leur généralisation en microinformatique, marquent une étape essentielle de l'évolution du développement d'applications pour PC, il est donc important, afin de concevoir des logiciels performants, de connaître les principes et concepts sous-jacents de cette technologie.

Le but du présent article est de rappeler ces principes et concepts. Il se termine par une présentation du SQL au travers de Delphi.

SQL: Strutured Query Language, Langage dInterrogation Structuré

Agenda

- Introduction
- Les SGBD-R
- La Normalisation
- SQL et DELPHI
- Conclusion

Introduction

Depuis sa première version 16 bits DELPHI intègre le BDE,
Moteur de base de données relationnel

BDE : Borland Database Engine

Le dialogue avec un SGBD-R passe par un langage : SQL

Historiquement, le BDE supporte aussi un autre langage : QBE

1

2

4

Les SGBD-R

- Le BDE, bref historique
- Concepts fondateurs
- Les 12 règles d ’intégrité du Dr Codd

Un moteur de base de données relationnel est avant tout un logiciel comme un autre. Sa fonction principale est de stocker de l'information et de la restituer en établissant un dialogue avec l'application cliente. Ce dialogue repose sur un langage, généralement SQL, dont nous verrons plus loin la nature.

Le BDE est un SGBD-R. Il a été originellement conçu pour gérer le format de données Paradox. A ce titre il privilégie, en interne, un autre langage d'interrogation qui est le QBE. Dans ses premières versions l'ancêtre du BDE était un logiciel DOS dit " résidant " avec lequel le développeur dialoguait par des Interruptions. Après le rachat de dBase, Borland a entrepris de fusionner les méthodes d'exploitation des données de ce produit et de celles de Paradox. Puis, au fil des évolutions, le BDE a pris corps, notamment en devenant une interface de programmation unifiée donnant accès à d'autres moteurs de base de données. Le support du SQL local a aussi été ajouté, en plus du QBE. C'est ainsi que le BDE s'est aussi ouvert aux données gérées via ODBC de Microsoft et aux principaux grands SGBD-R du marché via des pilotes spécialisés et optimisés que sont les SQL Links.

Originellement créé par ANSA Software puis acheté par Borland, aujourdhui vendu par Corell sous Licence Borland.

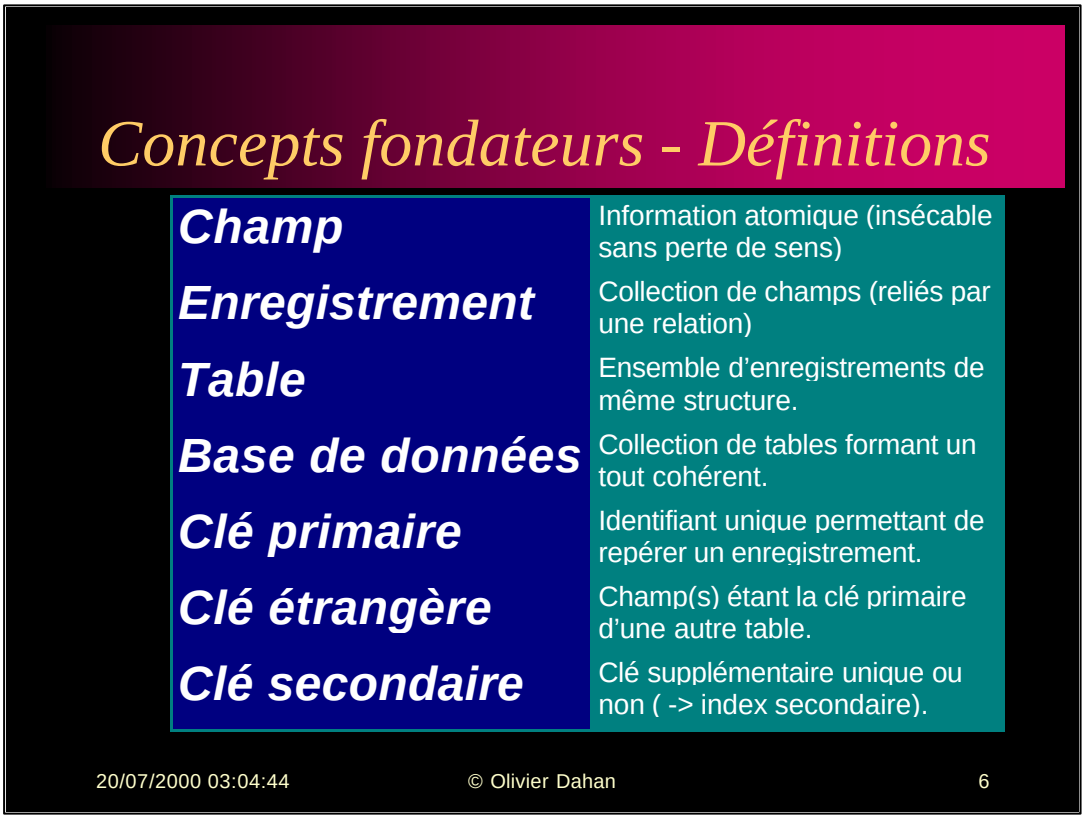
QBE: Query By Example, Requête par lexemple.

Technique de programmation utilisant des adresses mémoire, points dentrée, permettant un dialogue entre une application cliente (appelante) et une librairie de fonction (serveur). Le DOS se programmait de cette façon ainsi que tous les modules chargés en mémoire appelés résidants.

Aujourd'hui, sous Windows, le BDE se présente sous la forme d'une série de DLL proposant une interface de programmation (API) qui a été encapsulée dans Delphi sous la forme de composants.

API: Application Program Interface, interface de programmation.

Composant: Objet répondant à une syntaxe particulière lui permettant dêtre ajouté à la palette de Delphi.



Les SGBD-R utilisent une terminologie propre dont voici un extrait :

Champ Plus petit élément insécable d'information. Il peut être aussi petit qu'un Bit ou aussi grand qu'une image. Le contenu d'un champ ne peut être découpé en sous-unités plus petites sans perte de sens de l'information. Un champ ne peut avoir qu'une seule signification constante dans le temps.

Enregistrement Collection de champs dont l'unité repose sur une ou plusieurs re lations. Si on imagine un fichier de données sous la forme d'un tableau, les champs sont les colonnes, les enregistrements les lignes.

Table Collection d'enregistrements ayant la même structure (même découpage de champs). Souvent confondu avec un fichier, le concept de table est pourtant radicalement différent notamment par le fait qu'il peut fort bien n'y avoir aucun fichier représentant " physiquement " les données d'une table. De plus, le concept de table impose une structure (au moins logique) alors qu'un fichier peut contenir n'importe quoi, stocké n'importe comment.

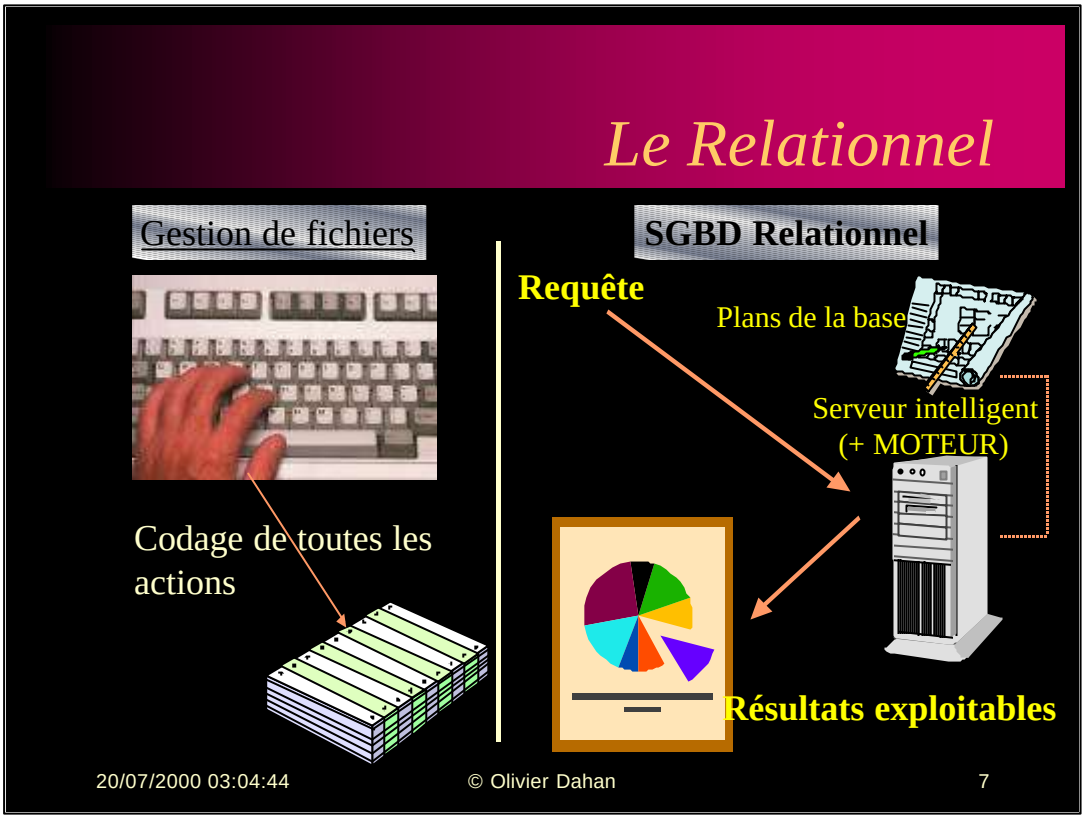
Base de données Collection de tables. Une base de données peut être aussi bien un sous-répertoire sur un disque (comme pour dBase ou Paradox) qu'une seule et même entité gérée par un Serveur local ou distant (Interbase, Oracle...).

Clé primaire Un champ ou un ensemble de champs permettant d'identifier sans ambiguïté tout enregistrement d'une table. Par essence les clés primaires sont uniques (elles n'acceptent pas les doublons).

Clé étrangère Un champ ou un ensemble de champs à l'intérieur d'une table qui se trouvent être la clé primaire d'une autre table.

Clé secondaire Une table peu posséder, en plus de sa clé primaire, d'autres clé s dites secondaires. Elles sont concrétisées par des index (secondaires eux aussi). Les clés secondaires se caractérisent par le fait que pour être clés elles sont uniques (à la différence d'un simple champ d'index secondaire).

Les clés secondaires ont un intérêt pratique mais ne font pas directement partie de la théorie relationnelle.



La caractéristique la plus marquante des SGBD-R ... c'est le " R "...

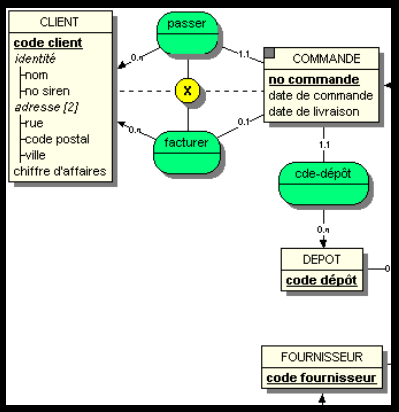
En effet, l'évolution majeure se trouve justement dans la capacité de ces systèmes à gérer des relations entre des entités et de maintenir l'intégrité de ces relations.

Dans une gestion de fichier classique, de type séquentiel indexé, c'est le développeur qui s'occupe de tout. Lorsqu'il veut interroger les données, il doit écrire une boucle pour balayer un ou plusieurs fichiers, parfois par une indirection depuis un ou plusieurs index, afin de trouver les données intéressantes et les fournir dans un ordre précis. Une telle programmation est incontournable puisque dans un tel cas il n'existe aucune " intelligence " sur laquelle le développeur puisse se reposer pour le soulager.

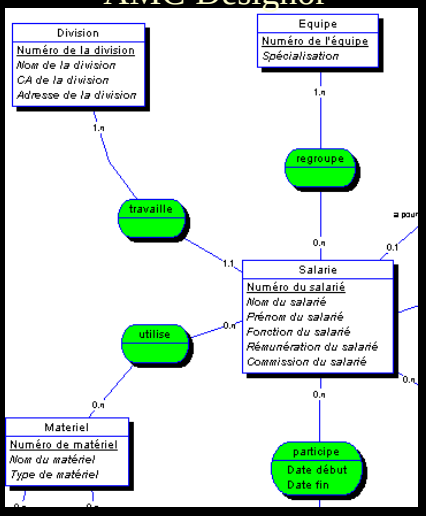
Outils CASE

Computer Assisted Software Engineering

Win-Design



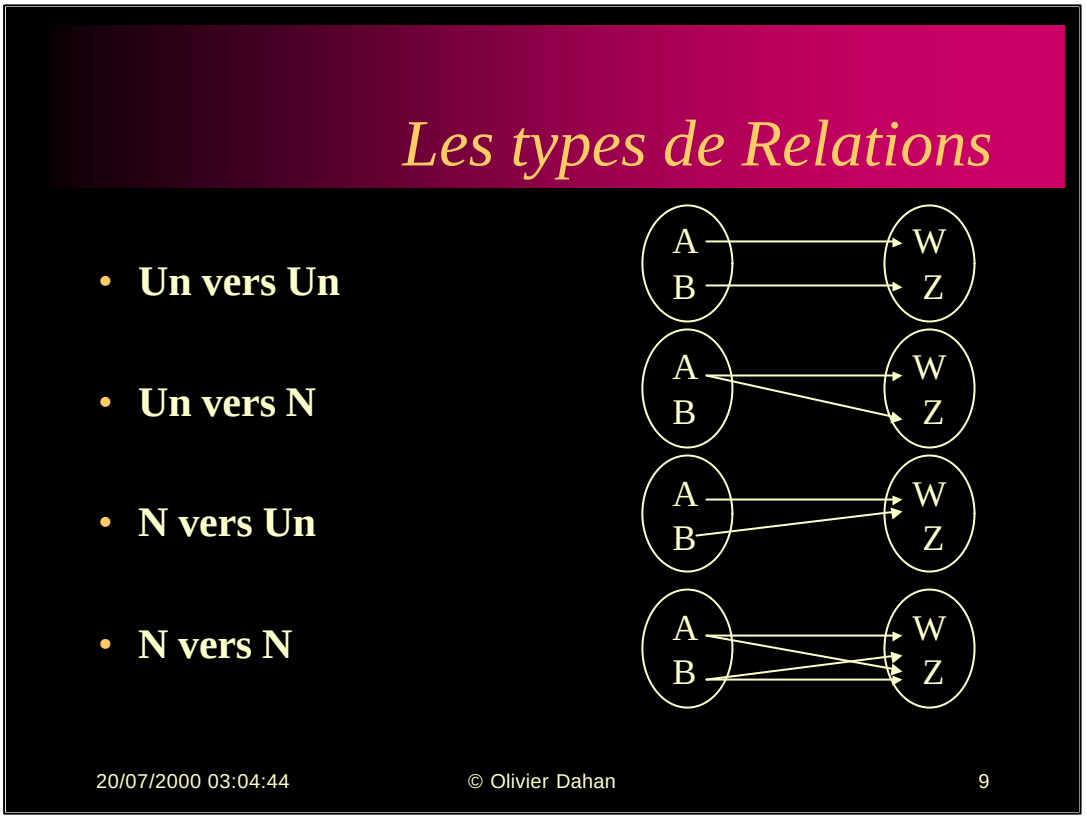
AMC Designor



20/07/2000 03:04:44 © Olivier Dahan 8

Les SGBD-R sont, comme nous le disions plus haut, des logiciels à part entière, dès lors ils proposent une certaine puissance, une “ intelligence ” qui prend en charge certains aspects essentiels de la gestion des données. De plus, un SGBD-R connaît les données qu’il gère : leur structure, les relations entre les champs et les tables. De fait, il est capable d’automatiser toutes les tâches de “ bas niveau ” (maintenir les index, gérer les conflits d’accès, autoriser les annulations de modification, etc) mais aussi d’assurer des fonctions de plus haut niveau comme la mise à jour par lot d’enregistrements répondant à des critères éventuellement complexes ou l’établissements de résultats synthétiques utilisant des calculs statistiques et des regroupements (par exemple le CA total et moyen par famille d’articles et par représentant sur une période donnée).

Un SGBD-R ne devine rien, il exploite les informations que le développeur lui a données. C’est lors de la conception même de la base de données que les relations entre les entités seront mises en évidence. La modélisation d’une base de données relationnel passe par une phase de conception durant laquelle on peut utiliser des outils CASE qui en simplifient la mise en œuvre, comme, par exemple, AMC Designor ou Win-Design.



Dans un modèle relationnel, on distingue plusieurs types de relations entre les entités. Lors du passage du modèle conceptuel (logique) au modèle Physique ces relations seront représentées soit par l'insertion de clé étrangères dans certaines tables, soit par la création de nouvelles tables. Voici les relations utilisées en pratique :

Le modèle conceptuel représente toutes les relations existantes entre les entités. Par exemple, il peut y avoir un entité Client et une entité Commande entre lesquelles il existe une relation Un-vers-N.

Le modèle physique est la concrétisation du modèle logique une fois la base cible choisie. Les entités deviennent des tables, les propriétés deviennent des champs dans les tables, et on voit apparaître les clés étrangères ainsi que les tables donnant corps à certaines relations multiples.

1.1.1.1 Relation Un-vers-Un

Une relation Un-vers-Un existe quand pour chaque clé primaire il y a zéro ou une valeur correspondante pour un autre champ ou un autre enregistrement. Lorsque la relation concerne des champs, ceux-ci sont généralement placés dans une même table, mais cela n'est pas obligatoire. Lorsque la relation concerne des entités (des enregistrements dans le modèle physique) elle peut s'entendre entre deux entités ou bien d'une entité vers elle-même. Ce dernier cas permet la représentation de hiérarchies (par exemple l'entité " employé " peut avoir une relation vers une autre entité " employé " avec la définition " a pour chef... ").

Un tel bouclage dune table sur elle-même s'appelle aussi une auto-jointure

1.1.1.2 Relation Un-vers-N

Une relation Un-vers-N (un vers plusieurs) existe quand, pour chaque enregistrement dans une table il existe zéro ou plusieurs enregistrements liés dans une autre table (exemple : un client a plusieurs commandes).

1.1.1.3 Relation N-vers-Un

Une telle relation existe lorsque plusieurs enregistrements d'une même table font référence de façon non ambiguë à un seul enregistrement d'une autre table. On appelle parfois cette relation une relation " Lookup " ou table de référence car elle représente une référence vers la clé primaire d'une autre table. Une relation N-vers-Un impliquant des clés étrangères peut généralement être renversée (en Un-vers-N), alors qu'une relation Un-vers-N impliquant que des clés primaires (liens identifiants) ne peut pas être renversée en N-vers-Un.

1.1.1.4 Relation N-vers-N

Une telle relation existe lorsque plusieurs enregistrements d'une même table sont liés à plusieurs autres enregistrements d'une autre table. Cette relation est réversible par essence. Un exemple classique est celui des commandes et des articles : plusieurs commandes peuvent faire références au même article, et plusieurs articles peut être utilisés dans une même commande. Les bases de données relationnelles ne peuvent représenter directement les relations N-vers-N, le concepteur de la base doit créer une table intermédiaire (contenant des couples formés des clés primaires des deux tables en relation plus d'éventuelles propriétés spécifiques à la relation). Si on utilise un outil de type AMC Designor ou Win-Design, c'est cet outil qui lors de la traduction du modèle logique en modèle physique fera apparaître la table intermédiaire automatiquement. Nous verrons un tel exemple plus loin.

0- Pour qu'un système de gestion de données soit considéré comme une base de donnée relationnelle, il doit utiliser exclusivement les relations pour gérer la base de données.

1- Toutes les informations de la base de données doivent être représentées d'une seule et même manière, notamment par des valeurs dans les colonnes à l'intérieur de lignes.

2- Chaque information stockée dans une base de données relationnelle doit pouvoir être accessible en précisant uniquement le nom de la table, le nom de la colonne et la valeur de la clé primaire.

3- Il doit y avoir un moyen d'exprimer des valeurs manquantes dans la base de données d'une façon homogène non dépendante du type de donnée, expression qui doit être supportée dans les opérations logiques. La règle originale précise qu'il faut différencier valeur manquante et valeur inapplicable, à l'heure actuelle les SGBD-R ne proposent majoritairement que le NULL.

4- La description de la base au niveau logique doit être représentée dynamiquement par un ensemble de données normalisé de telle sorte que ce catalogue soit accessible aux utilisateurs autorisés depuis leur langage de requête.

On voit ici que les formats dBase et Paradox ne satisfont pas à cette exigence alors que Interbase ou tout autre serveur SQL ne saurait fonctionner sans tables système. Ces pour cela qu'on choisira les seconds si on doit gérer beaucoup d'intégrité référentielle ou de contraintes.

5- Le système doit mettre à la disposition au moins un langage relationnel qui : (1) a une syntaxe linéaire, (2) peut être utilisé à la fois de manière interactive et dans des programmes d'application, et (3) supporte les opérations de définition de données (dont les vues), celles de manipulation de données (mise à jour et accès), les contraintes de sécurité et d'intégrité, ainsi que les opérations de pilotage de transactions (début, validation, annulation). Le langage SQL propose tout cela.

6- Le système doit supporter la définition des vues et les informations des tables sous-jacentes doivent pouvoir être mises à jour directement depuis la vue. Les vues modifiables ne sont pas gérées de la même façon par tous les serveurs, et les vues, même non modifiables n'existent pas en dBase et Paradox.

7- Le système doit permettre aux opérateurs INSERT, UPDATE et DELETE d'être actifs en même temps. Le SQL standard supporte cette contrainte.

8- Les programmes d'application et les opérations interactives ne doivent pas avoir à être modifiés si la structure physique de la base est modifiée. En ce sens SQL est un langage beaucoup plus portable que la majorité des langages classiques. Delphi et le BDE autorise d'ailleurs une très bonne isolation entre représentation physique de la base et application. On peut à ce sujet étudier l'exemple fourni avec Delphi (MASTAPP) qui fonctionne aussi bien avec des données Paradox que sous Interbase.

9- Les programmes d'application et les opérations interactives n'ont pas être modifiés si la base de données est restructurée, du moins tant qu'il n'y a pas de perte d'information.

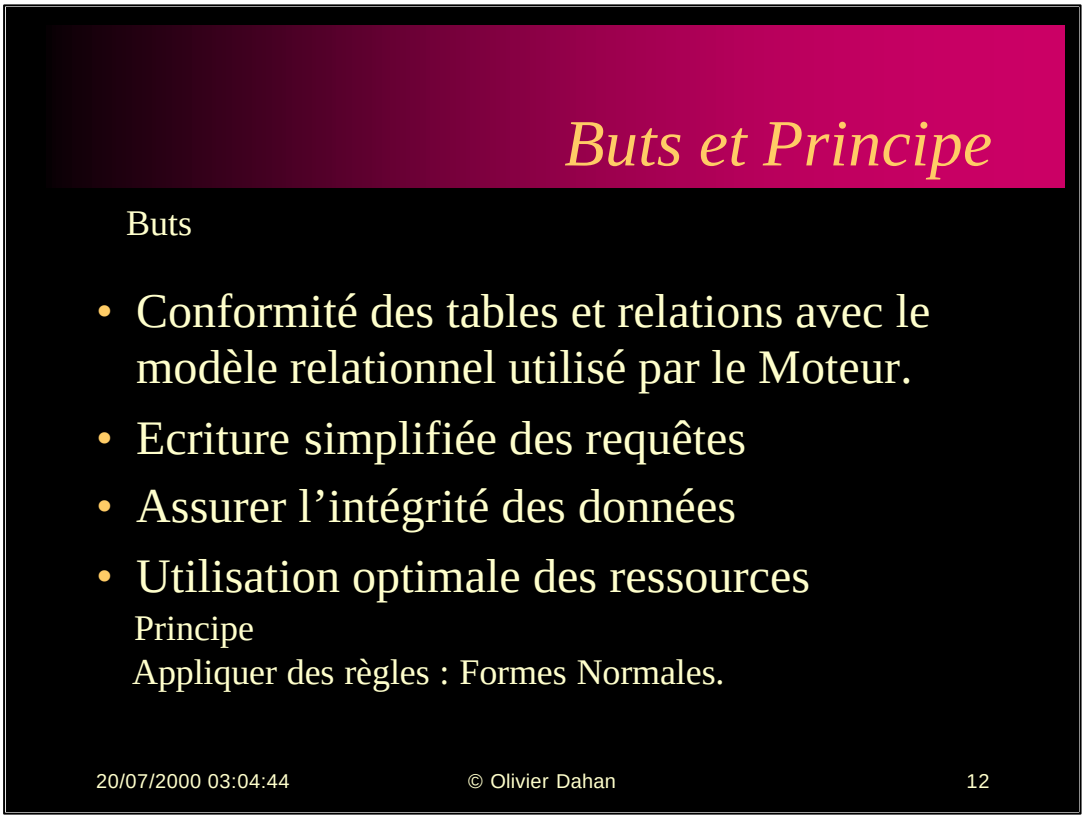
10- Les contraintes d'intégrité doivent être spécifiées séparément des programmes d'application et rangées dans le catalogue. On doit pouvoir les modifier comme on le souhaite, sans avoir à toucher aux applications existantes.

11- Les applications existantes se dérouleront normalement (1) quand une version distribuée du SGBD est introduite pour la première fois et (2) quand les données distribuées sont redistribuées dans le système. Les versions distribuées de SQL commencent tout juste à apparaître et il est difficile aujourd'hui de donner des exemples précis de respect de cette règle.

12- Si le système dispose d'une interface de bas niveau (traitement d'un enregistrement à la fois), celle-ci n'a pas la possibilité de pervertir le système, c'est à dire de passer outre une directive de sécurité relationnelle ou une contrainte d'intégrité. SQL-92 répond à ces conditions.

La Normalisation

- But et Principe
- Dépendances fonctionnelles et Dépendance de plusieurs valeurs
- Les Formes Normales (NF)



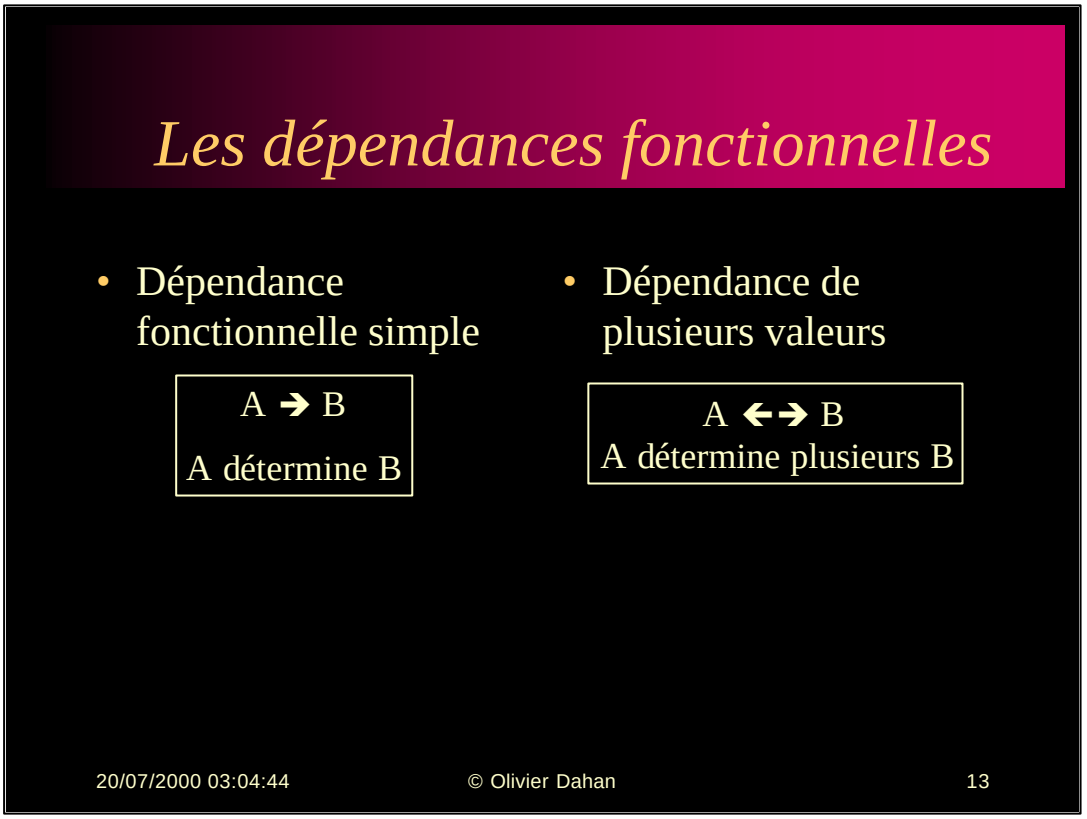
Il y a plusieurs bonnes raisons de normaliser une base de données, la première relève uniquement du bon sens : Le BDE, et tous les serveurs SQL, ont été conçus sur la base de l'hypothèse que les données qu'ils auraient a traiter seraient normalisées. De ce fait, ces SGBD-R sont plus efficaces lorsque la base utilisée est normalisée. Cela peut paraître une évidence, mais tout va mieux lorsqu'on le dit...

Parmi les raisons plus spécifiquement liées au relationnel, la normalisation apporte : des requêtes plus simples à écrire*, des données plus facilement accessibles ; une meilleure intégrité des données ; la diminution des erreurs lors de l'insertion ou de la suppression de nouvelles données et une utilisation optimale des ressources.

*Mais pas forcément plus courtes!

Il faut tout de même préciser que la normalisation des bases de données n'est pas une fin en soi, seulement un outil pratique et performant. Chaque concepteur de base de données doit décider si, dans un cas précis, la normalisation est la solution la plus efficace.

La dénormalisation est une opération parfois nécessaire, toutefois il faut bien l'entendre dans le sens que son nom indique : la suppression d'une normalisation existante... Il est donc toujours préférable, au niveau conceptuel, de commencer par concevoir une base de données normalisée. Ensuite, et seulement ensuite, on peut dénormaliser en justifiant et en assumant chaque opération ponctuelle de ce type.



Une forme normale (que nous aborderons à la section suivante) est une méthode de classification de table qui repose sur les dépendances fonctionnelles (DF) qu'elle comprend. Une dépendance fonctionnelle signifie que si l'on connaît la valeur d'un attribut on peut toujours déterminer la valeur d'un autre attribut. La notation utilisée dans la théorie relationnelle est une flèche entre les deux attributs, par exemple A $\rightarrow$ B s'énonce " A détermine B ". Si on connaît votre numéro de salarié dans l'entreprise on peut trouver votre nom.

La dépendance de plusieurs valeurs (DPV) signifie que si l'on connaît la valeur d'un attribut on peut toujours déterminer les valeurs d'un ensemble d'autres attributs. La notation retenue dans la théorie relationnelle est une flèche à double pointe entre les deux attributs, par exemple A $\leftrightarrow$ B s'énonce " A détermine plusieurs B ". Si on connaît le nom d'un professeur on peut déterminer la liste de ses étudiants.

La compréhension des DF et les DPV joue un rôle essentiel dans celle des formes normales.

Les Formes Normales

Ce sont des règles à appliquer :

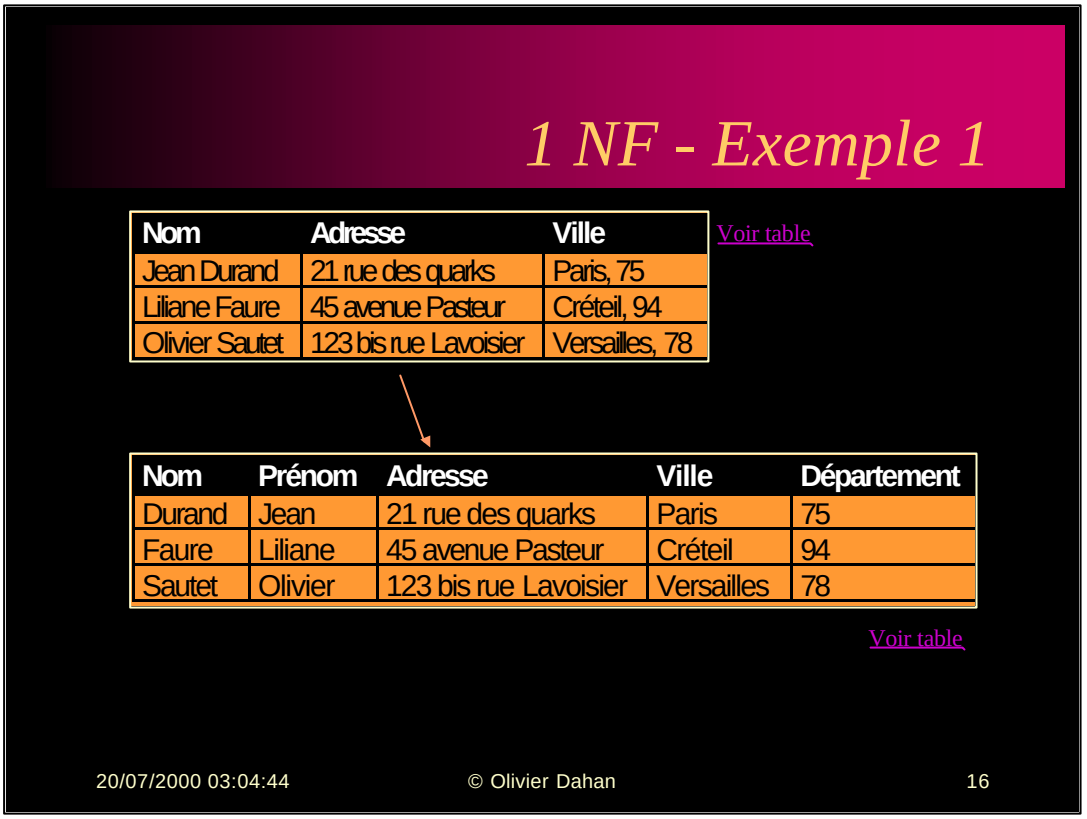
- A la conception d'une base pour s'assurer de sa cohérence.
- Sur une base existante pour en vérifier la cohérence.

Normaliser une base consiste à appliquer des règles regroupées sous la dénomination de " Formes normales ". Ces règles sont aussi utilisées, dans l'autre sens, pour valider l'état de normalisation d'une base. Les formes normales sous entendent que chaque table possède une clé primaire. Chaque forme porte un numéro d'ordre (ou un acronyme)*. Les voici :

*Normal Form en anglais, noté NF précédé du numéro de la forme comme 1NF pour la 1ère forme normale.

Il s'agit de la première règle qu'on pourrait dire fondatrice. Elle fait partie de la définition formelle des bases de données relationnelles.

Cette règle stipule que les champs de chaque table doivent être atomiques et qu'il ne peut exister de champs répétitifs. De plus, chaque champ doit avoir une signification précise constante dans le temps.

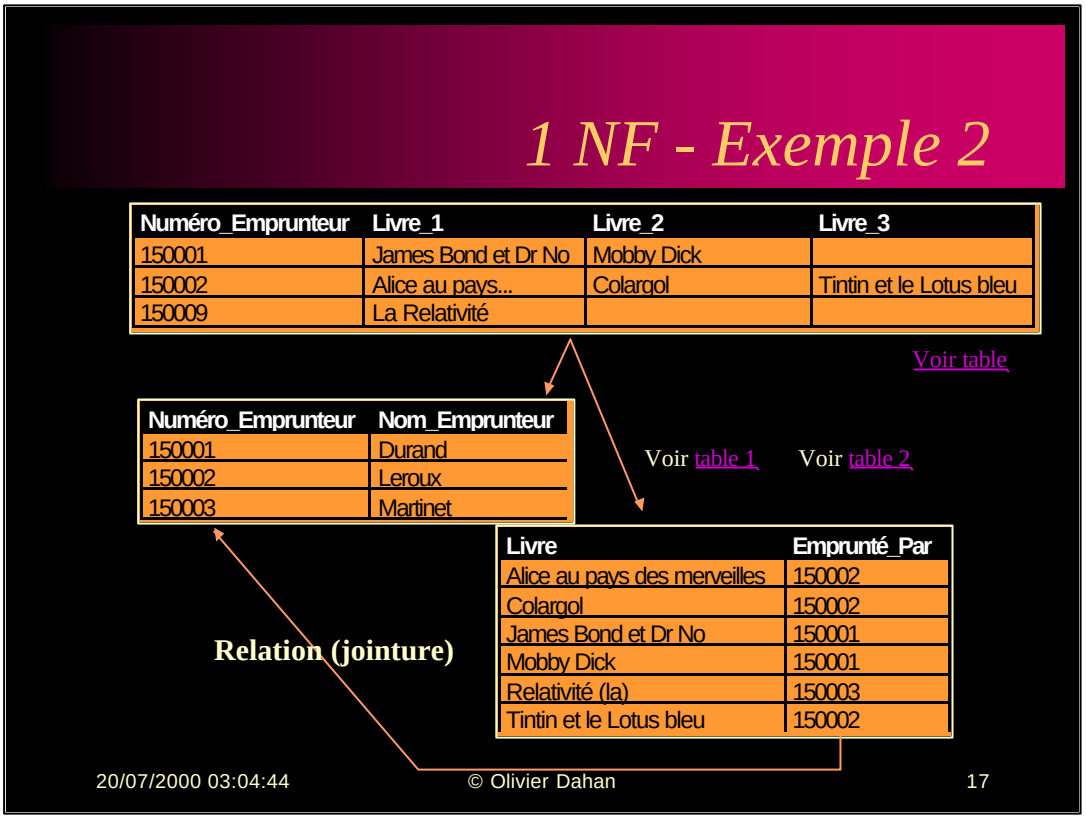


Pourquoi cette table n'est-elle pas conforme à la 1ère forme normale ?

La raison principale est que les champs ne sont pas tous atomiques. Par exemple le champ " Nom " contient à la fois nom et prénom, il est possible de découper ce champ sans faire perdre de sens à l'information, ce qui en faciliterait même l'accès et le tri. De la même façon, le champ " ville " intègre le numéro de département, ce qui n'est pas très pratique si on désire sortir, par exemple, la liste de toutes les personnes du Val de Marne. Dans un tel cas il faudra analyser chaque enregistrement en espérant que la saisie est homogène (que la virgule est toujours présente, qu'il n'y a pas de caractères non valides, etc).

En fait, pour faire que cette table réponde à la 1ère forme normale, il faudra la restructurer pour qu'elle ressemble à la seconde image.

Maintenant que tous les champs sont atomiques, les tris et les recherches deviennent plus simples, plus directs. Certaines personnes préconisent de découper aussi l'adresse en numéro de rue, rue, etc... En fait il existe une norme ISO des adresses qu'on est libre d'utiliser ou non. La variété des types de voies, des bourgades et autres lieux-dits rend un tel découpage lourd et peu adapté à la majorité des applications de gestion. Il ne nous a jamais été donné de voir un cadre d'application dans lequel une telle opération avait une raison fonctionnelle, mais cela existe.



En quoi cette table ne répond-t-elle pas à la 1ère forme normale ?

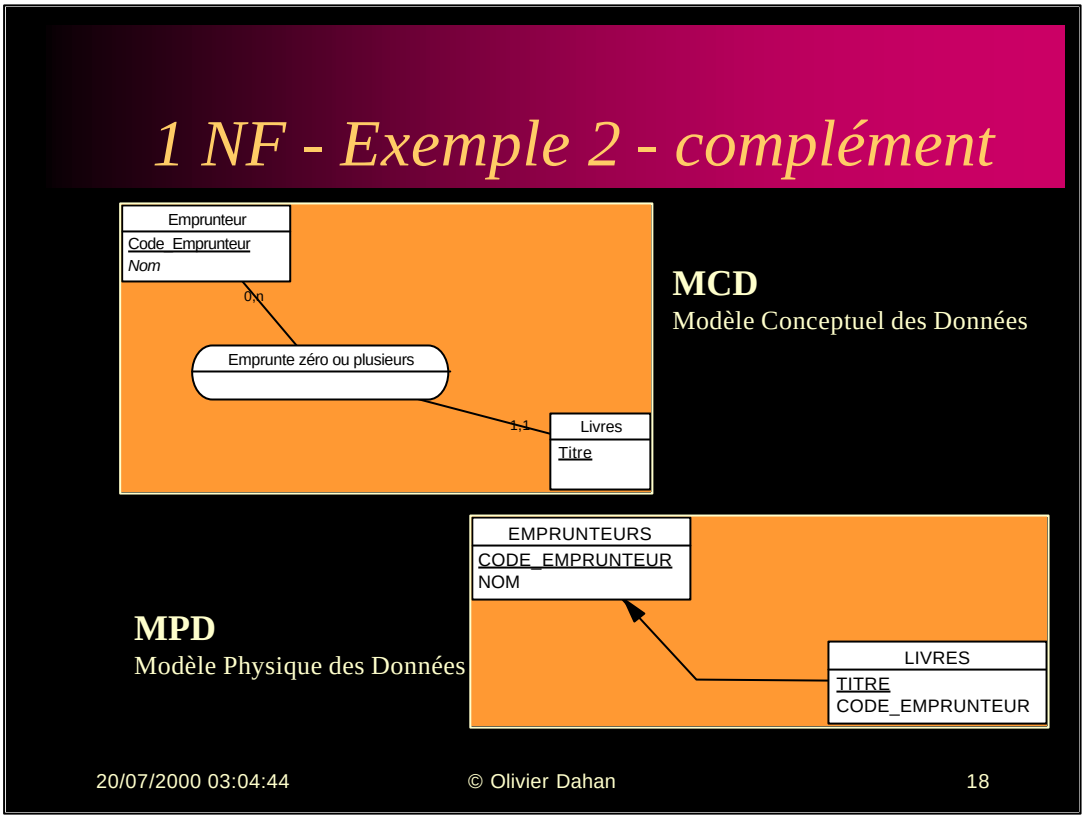
La réponse se trouve dans la liste des livres.

En effet, la 1ère forme normale stipule que les champs ne peuvent pas être répétitifs. De fait, on comprend que chercher qui a emprunté " Tintin et le Lotus bleu " obligera à balayer chaque enregistrement puis à rechercher la présence de ce titre dans chacune des trois colonnes.

Comme dans le 1er exemple, il faudra espérer que la saisie est homogène et que ce titre aura bien été tapé de la même façon que celui qu'on recherche, le moindre espace de trop fera échouer la recherche...

Plus loin, si demain il faut gérer 4 livres au lieu de trois cela impliquera à la fois une modification de la table (intervention d'un spécialiste sur chaque site ou création d'un logiciel de mise à jour qu'il faudra tester et packager) et une refonte de toutes les séquences de recherche qui réclameront une fois encore des tests ainsi que la fourniture d'une mise à jour du logiciel à chaque site utilisateur. Autant de dépenses qui peuvent être très lourdes, uniquement pour avoir négliger de normaliser la base de données... On voit ici que l'intérêt de cette démarche va bien au-delà de la satisfaction intellectuelle.

Afin de rendre la table conforme à la 1ère forme normale il nous faudra la scinder au minimum en deux autres tables, voire en trois.



Le choix repose ici sur les cardinalités de la relation.

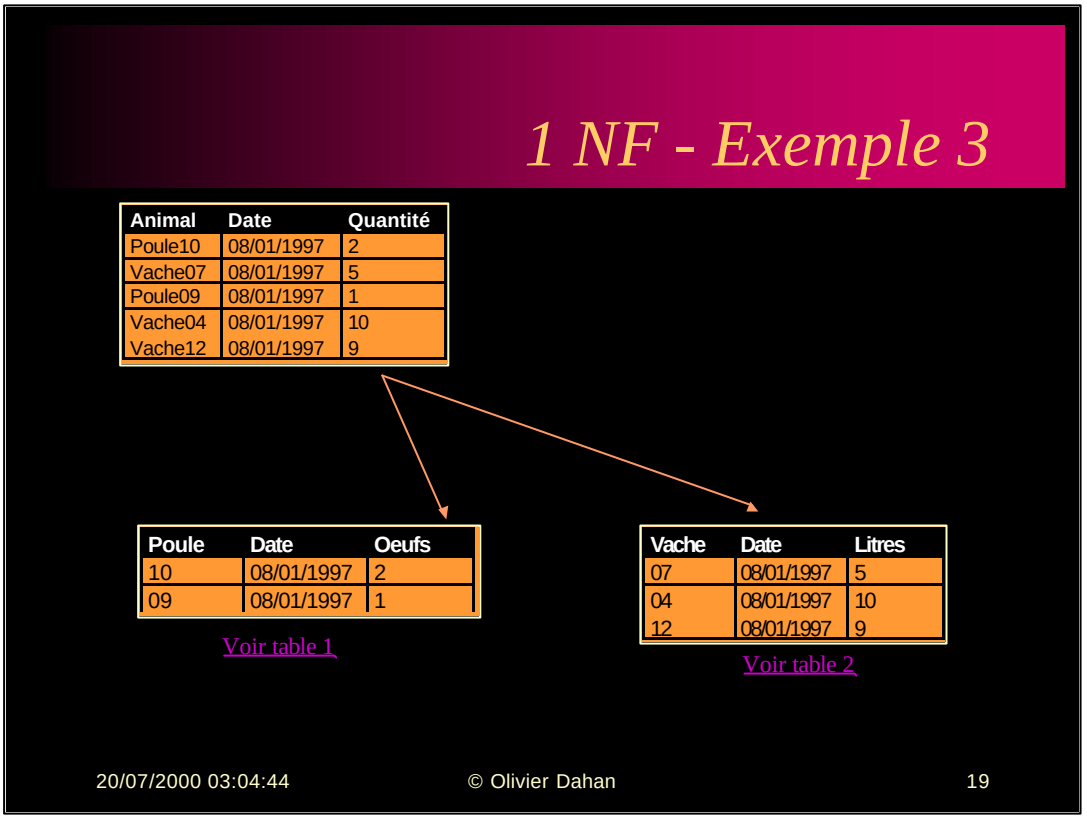
Les cardinalité dune relation sont les nombres qui caractérisent le type de relation discutés à la section .

S'agissant de livres et s'il n'existe qu'un exemplaire de chaque titre, il ne pourra s'agir, entre les emprunteurs et les livres que d'une relation Un-vers-N car une même personne peut emprunter plusieurs livres alors qu'un livre ne peut être emprunté que par une seule personne à la fois.

S'il s'agissait non plus d'emprunteurs et de livres mais d'élèves affectés à des cours, la relation serait de type N-vers-N, un élève pouvant suivre plusieurs cours et un même cours pouvant recevoir plusieurs élèves (dans ce cas trois tables seraient nécessaires). Ce cas pourrait être aussi celui des livres s'il existe plusieurs exemplaires des mêmes titres.

Sous cette nouvelle forme on voit qu'il est fort simple de savoir immédiatement qui a emprunté quel livre ou de connaître la liste des livres empruntés par une personne. Dans le premier cas il suffira de lire le code Emprunteur dans la table des livres puis d'accéder à l'enregistrement de celui-ci pour en connaître le détail. Dans le second cas il suffira de balayer la colonne des codes emprunteurs de la table des livres pour connaître tous les livres empruntés par une même personne. En fait, de telles opérations sur une base de données normalisée s'effectuent de façon fort simple à l'aide du SQL.

Il faut noter qu'on suppose qu'ici le champ " Numéro_Emprunteur " est la clé primaire de la table des emprunteurs et que le champ " Livre " est la clé primaire de la table des livres. De ce fait, le champ " Emprunté_par " de la table des livres est une clé étrangère.

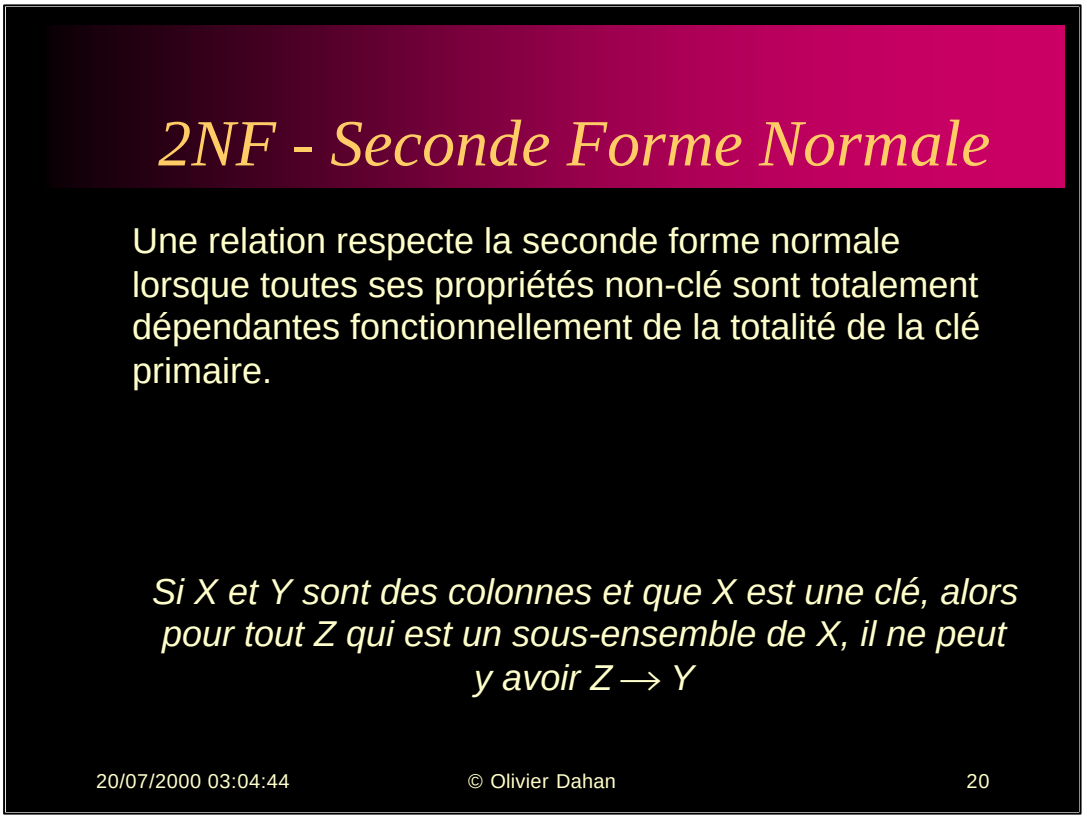


A première vue tous les champs sont atomiques. Il n'existe pas de champs répétitifs... et pourtant...

Il sera très difficile de faire des statistiques de production par jour, par exemple, car le champ " Quantité " représente une fois un nombre d'œufs, et d'autres fois des litres de lait.

En programmation structurée, si on exploite les enregistrements avec variantes on peut très bien se satisfaire d'une stockage de ce type qui aura, dans un tel cadre, l'avantage d'être compact (un seul fichier). Toutefois cet avantage n'a plus de sens dans le contexte des SGBD-R, et, au contraire, il interdit de se servir des ordres SQL qui permettront d'obtenir sans programmation réelle des résultats qui nécessitaient, avant, des algorithmes spécifiques.

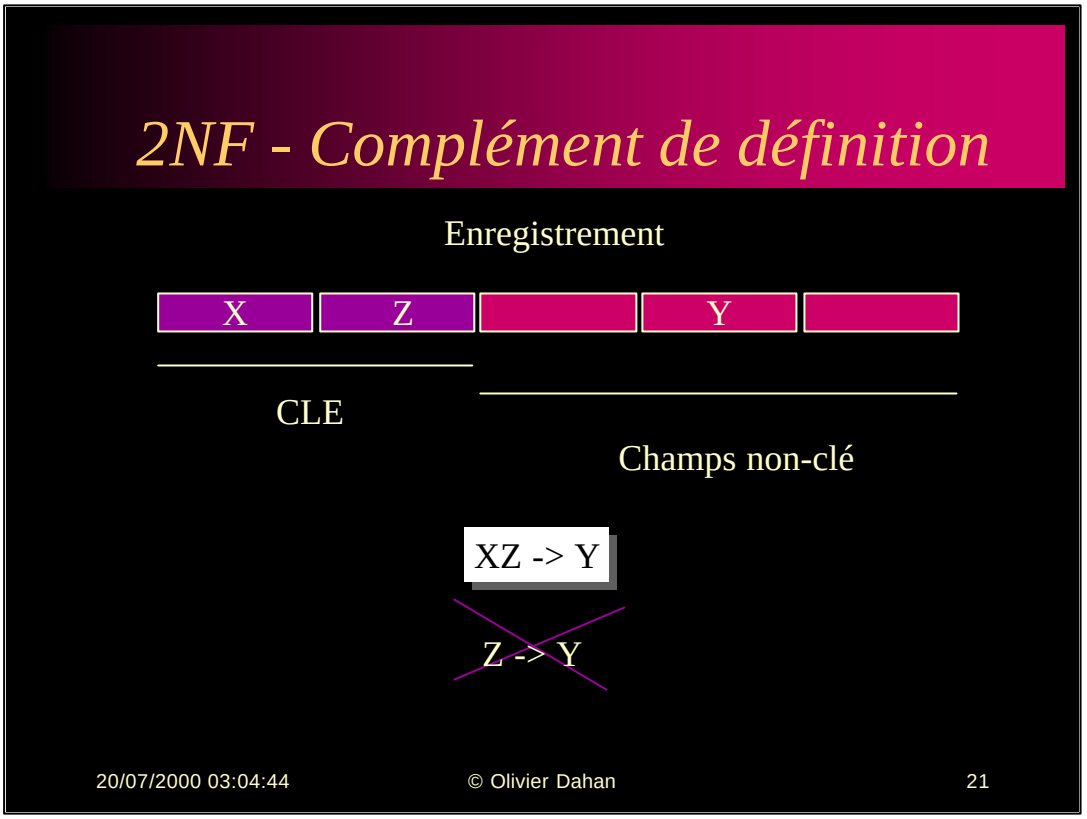
Ainsi, dans l'exemple ci-dessus, le champ " Quantité " n'a pas une signification précise constante dans le temps.C'est une violation de la 1NF. Il faudra restructurer les données.



Le respect de la seconde forme normale est aussi d'une grande importante. La définition qui suit vous semblera peut-être trop formelle, mais les exemples vous prouveront qu'ici aussi il ne s'agit que de bon sens.

Une relation respecte la seconde forme normale lorsque toutes ses propriétés non-clé sont totalement dépendantes fonctionnellement de la totalité de la clé primaire.

Si X et Y sont des colonnes et que X est une clé, alors pour tout Z qui est un sous-ensemble de X, il ne peut y avoir $Z \rightarrow Y$.

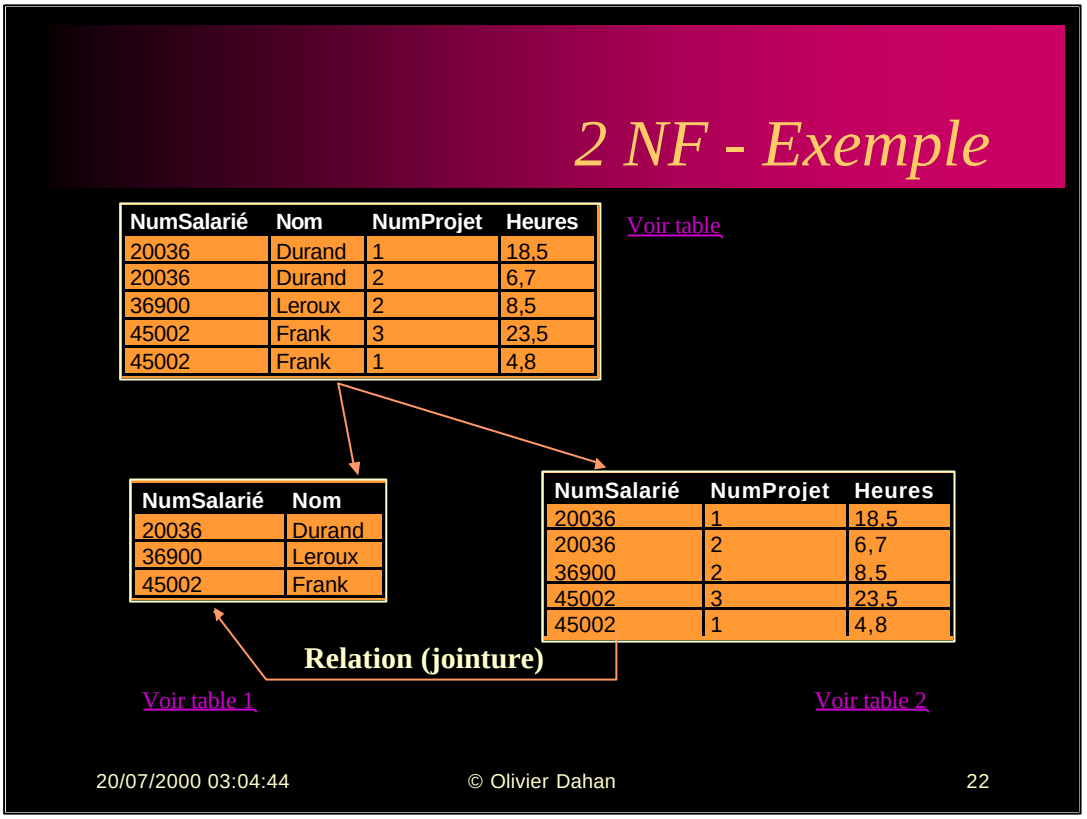


La clé est ici formée par deux champs : X et Y

L ’enregistrement contient d ’autres champs dont Y

La 2NF oblige que Y soit déterminé par le couple XY

Si seulement Z (ou X) permet de déterminer Y, la table n ’est pas dans la 2de forme normale.



Bien que cette table respecte la 1ère forme normale, elle ne respecte par le seconde.

Dans un premier temps nous devons déterminer où se trouve la clé primaire. Selon les critères même d'une clé primaire et selon le bons sens, il ne peut y avoir que quelques combinaisons permettant de repérer de façon unique chaque ligne ayant une signification : Numéro de salarié + Numéro de projet ou bien Nom + Numéro de projet. Selon l'utilisation qui sera faite de la table on peut renverser les clés, ce qui ne change rien à leur nature.

Choisissons la première solution (nous verrons que cela ne change rien) : Numéro de salarié + Numéro de projet.

Maintenant que la clé primaire a été définie, regardons les champs non-clé : Nom et Heures. Les heures sont en totale dépendance fonctionnelle avec la totalité de la clé puisque cette propriété concerne à la fois un individu et un projet et que seule cette combinaison permet d'isoler un compte d'heures unique.

Le nom du salarié, d'un autre côté, ne dépend pas de la totalité de la clé primaire. En fait il ne dépend que d'un morceau de celle-ci, le numéro de salarié. On s'aperçoit ici que si nous avions choisi l'autre possibilité pour la clé primaire nous aurions le même problème avec le numéro de salarié qui ne dépendrait que du nom et non de la totalité de clé.

Certains esprit chagrins feront sans doute remarquer que cette normalisation a créé la duplication de la colonne numéro de salarié et qu'il s'agit d'une perte de place.

Nous ferons alors remarquer que sous sa forme non normalisée la table obligeait la répétition du nom du salarié dans chaque enregistrement ce qui, s'agissant d'un texte et non d'un code, consomme une place très importante. De fait, la répétition du code salarié, permet au contraire de gagner beaucoup de place dans la base...

Dans la version normalisée on s'aperçoit rapidement que la destruction d'une enregistrement prend une signification bien précise selon qu'on se trouve dans une table ou dans l'autre. La suppression de tous les cumuls d'heures d'un salarié ne détruit pas les informations le concernant et qui n'ont aucun rapport avec les cumuls.

3 NF - 3ème Forme Normale

- Une relation est dite dans la troisième forme normale si aucun champ non-clé n'est en dépendance transitive avec la clé primaire.
- La BCNF (Boyce Codd NF) impose que la 3NF soit vérifiée pour toutes les clés primaires potentielles.

soit trois colonnes (A, B, C), A étant la clé primaire, si (A → B) et que (B → C) on peut en déduire que (A → C), dans ce cas il existe une relation transitive entre A et C et la table n'est pas dans la 3NF

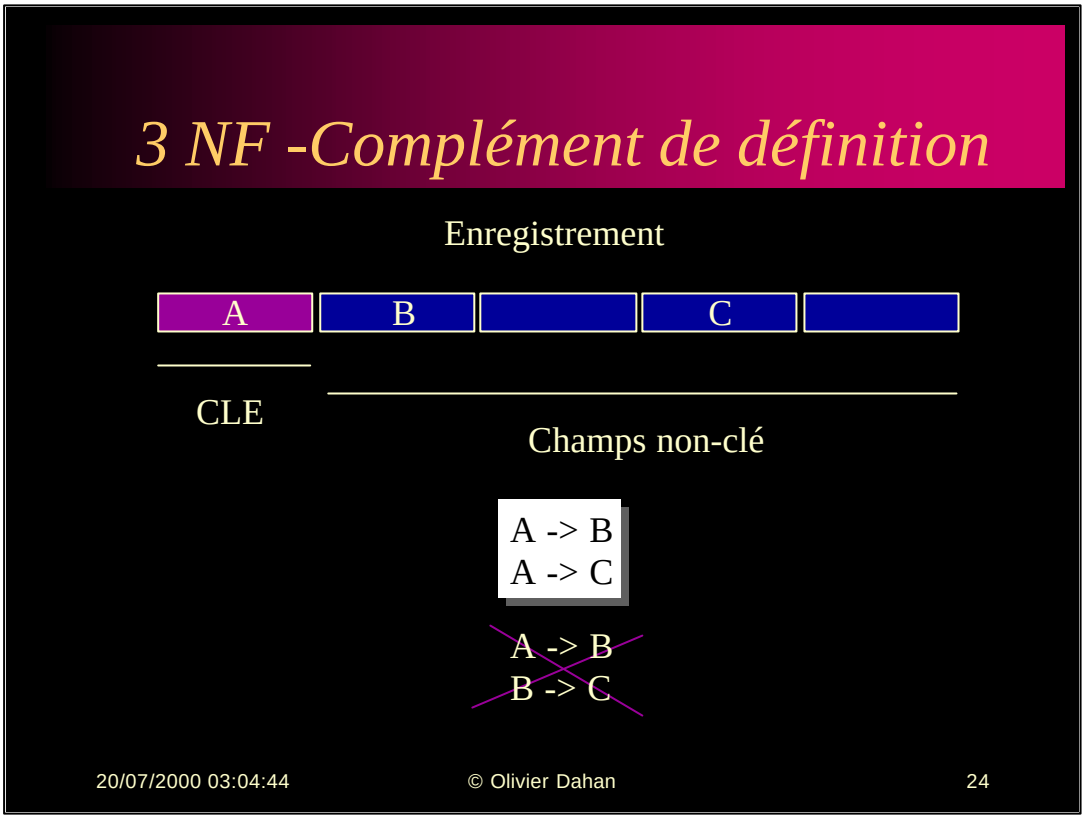
Voici sa définition :

Une relation est dite dans la troisième forme normale si aucun champ non-clé n'est en dépendance transitive avec la clé primaire.

Ce qu'on peut noter aussi : soit trois colonnes (A, B, C), A étant la clé primaire, si (A → B) et que (B → C) on peut en déduire que (A → C), dans ce cas il existe une relation transitive entre A et C et la table n'est pas dans la 3NF.

En fait il s'agit toujours de bon sens et cette définition peut se comprendre facilement : si la valeur d'un champ non-clé peut être déduite de la valeur d'une autre champ non-clé alors sa relation à la clé primaire est, par force, transitive (puisqu'elle transite par un autre champ) et la table n'est pas dans la troisième forme normale.

Voir la 3ème forme normale comme une extension de la 2de est une erreur que la trivialité des exemples proposés ici peut induire : la nature même des exigences est totalement différente.



Supposons que la clé primaire soit le champ A

L ’enregistrement contient d ’autres champs, dont B et C

Si la relation est normalisée on a :

A détermine B

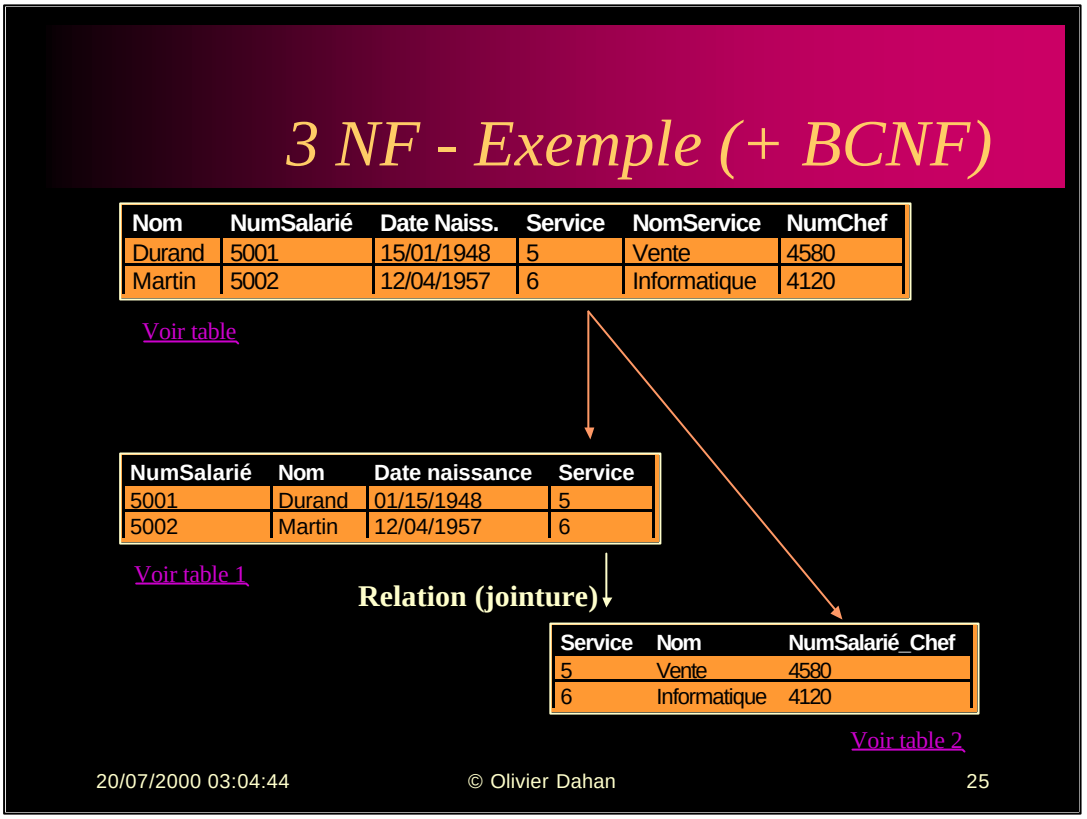
A détermine C

Si on trouve :

A détermine B

B détermine C

Le lien entre A et C est transitif et la relation n ’est pas normalisée.



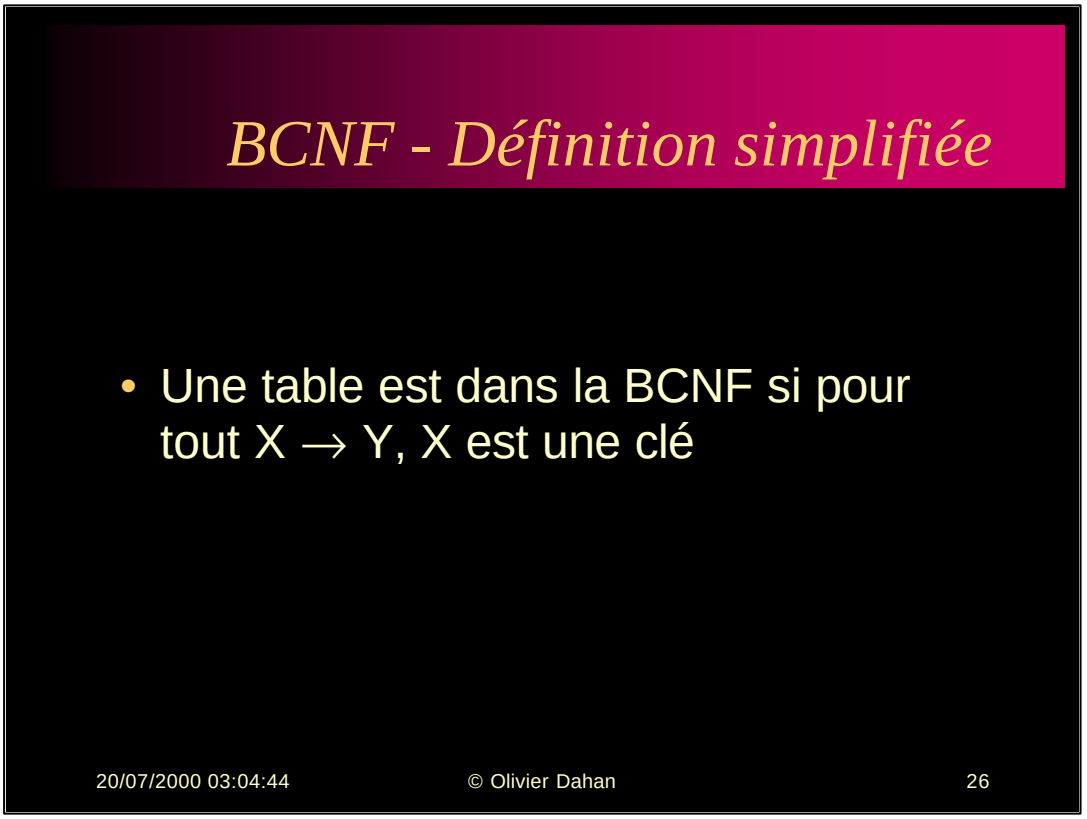
Cette table est dans la première forme normale (champs atomiques, non répétitifs, ayant une signification unique constante dans le temps, présence d'une clé primaire - par exemple ici le numéro de salarié), elle répond aussi aux exigences de la seconde forme normale (tous les champs non-clé sont en dépendance fonctionnelle avec la totalité de la clé primaire). Toutefois, elle n'est pas dans la troisième forme normale.

En effet, le nom du service ainsi que le numéro de salarié du chef de service sont liés à la clé primaire par une relation transitive qui passe par le code du service. Il est effectivement possible de déterminer le nom du service et le code salarié de son chef uniquement à partir du code service.

Quel problème l'état actuel va-t-il poser ?

Cela se rapproche beaucoup de celui posé dans l'exemple fourni pour le seconde forme normale : si nous supprimons tous les employés d'une service donné, lors de la suppression du dernier enregistrement nous perdrons irrémédiablement dans le même temps les informations concernant le service lui-même (son nom et son chef). Cela est connu sous l'appellation *d'anomalie de suppression*.

De la même façon, si on créé un nouveau service dans l'entreprise nous ne pourrons pas l'ajouter tant qu'il n'y aura pas un salarié affecté à ce service. Ce problème est connu sous le nom *d'anomalie d'insertion*.



Cette forme normale est une extension de la troisième forme, elle seule supprime toute dépendance transitive. Si la table possède plus d'un candidat pour la clé primaire, elle doit être examinée selon le point de vue de chacune de ces clés potentielles. Si après un tel examen elle se trouve toujours être dans la troisième forme normale (quel que soit la clé primaire choisie) alors elle est dans la forme normale de Boyce-Codd.

L'expression de la BCNF est d'une simplicité redoutable : Une table est dans la BCNF si pour tout X → Y, X est une clé. C'est tout !

Encore et toujours, il ne s'agit que de bon sens...

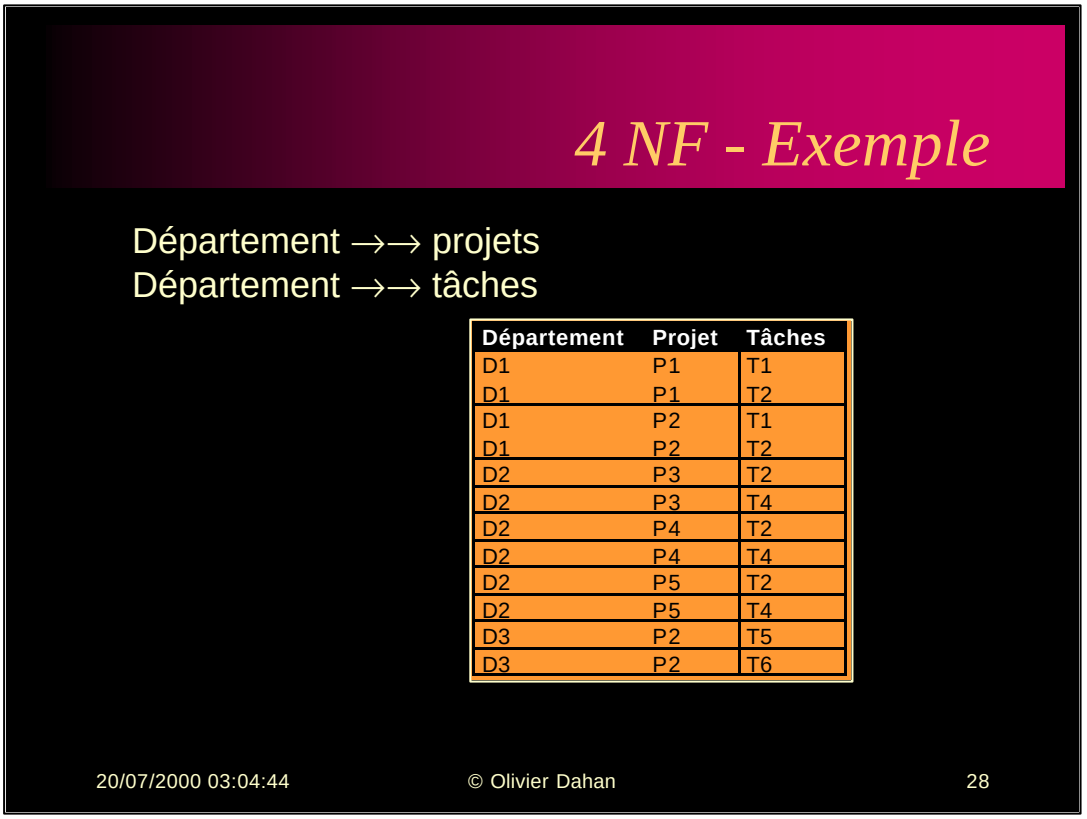
L'exemple pris pour la 2NF posait le cas de choisir la clé primaire qui pouvait être construite soit à l'aide du nom soit à l'aide du code du salarié. Pour que les tables (corrigées) soient considérées en BCNF il faut les tester jusqu'à la 3NF pour chacune de ces clés possibles.

Au risque d'une certaine répétition, rappelons que les exemples ici fournis sont particulièrement triviaux. Dans la réalité être certain qu'on a bien identifié toutes les clés primaires possibles n'est pas toujours évident.

4 NF - 4ème Forme Normale

- La 4NF règle les problèmes liées aux DPV trop nombreuses.
  (DPV: Dépendance de Plusieurs Valeurs)

La définition formelle de 4NF étant peu parlante, nous allons l ’étudier par l ’exemple qui suit...



Le plus simple est de prendre un exemple...

Supposons une table des départements d'une entreprise, de leurs projets (ou travaux), et des tâches qui leur incombent avec les dépendances de plusieurs valeurs (DPV) suivantes :

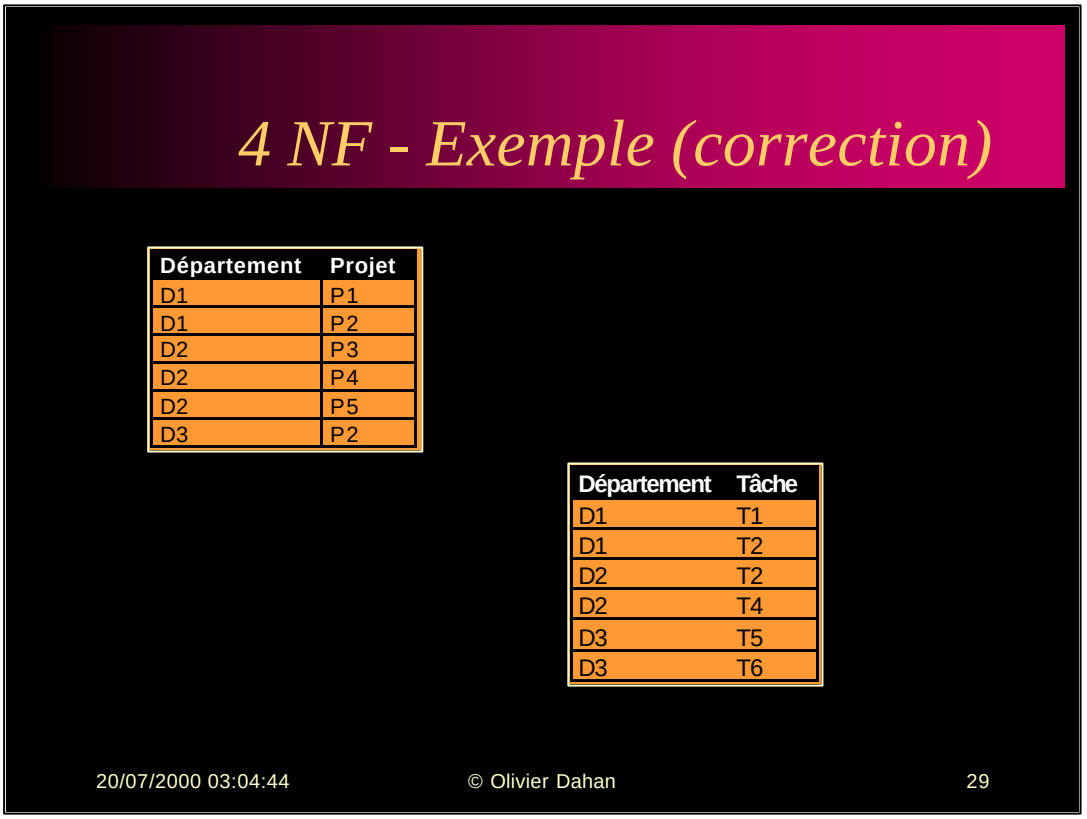
Département →→ projets

Département →→ tâches

supposons que le département D1 s'occupe des projets P1 et P2 et des tâches T1 et T2, le département D2 des projets P3, P4 et P5 et des tâches T2 et T4 et le département D3 du projet P2 seulement et des tâches T5 et T6. La table ressemblera à l 'image ci-dessus.

Si on veut ajouter une tâche à un département, il faut créer plusieurs lignes nouvelles (puisqu'il faudra générer et maintenir toutes les combinaisons possibles avec les projets). On risque ainsi, en supprimant une tâche ou un projet d'une ligne de perdre des informations. Pour mettre à jour le nom d'une tâche ou d'un projet il faudra aussi répéter l'opération sur plusieurs lignes.

La solution consiste à répartir les données en deux tables l'une établie d'après (Département, Projets) et l'autre d'après (Départements, Tâches). On peut en fait simplifier l'expression de la 4NF en disant qu'il ne faut conserver qu'une seule DPV par table.



En fait, partir de l'aspect physique des tables pour effectuer la normalisation n'est pas forcément une démarche très " académique ". L'exemple que nous avons pris simplifie certes la compréhension de la 4NF mais dans la réalité, et si nous étions parti du modèle conceptuel " propre " nous n'aurions obtenu ni la table de l'exemple ni même les deux tables du corrigé.

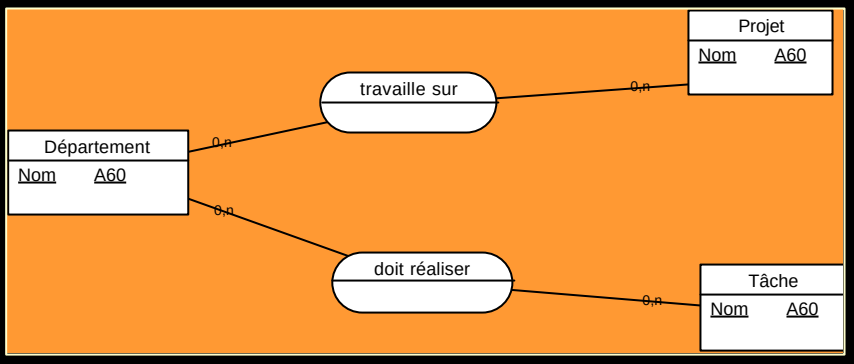
Cela est important car il faut comprendre que si on choisit de travailler " proprement " dès le départ, notamment en utilisant des outils CASE comme AMC Designor ou Win-Design, on élimine beaucoup de problèmes de dénormalisation car la notation Entité-Relation finit par imposer sa propre logique.

Dans notre exemple il faudra s'occuper dès le départ des cardinalités, notamment savoir si un projet ou une tâche peuvent être partagés par plusieurs département ou bien si une tâche ou un projet sont toujours affectés à un seul et unique département (on peut aussi envisager les combinaisons mixtes). De même est-il essentiel de savoir si tâches et projets sont totalement indépendants ou s 'il existe une relation.

On voit que ces questions force l 'étude de la sémantique de l 'application. Seule l 'analyse permet de répondre à ces questions. Créer des MCD lors de l 'analyse permet ainsi de poser des questions pertinentes auxquelles on n 'aurait pas pensé ...

4 NF - MCD exemple

1er cas : les projets et les tâches peuvent être partagés par plusieurs départements



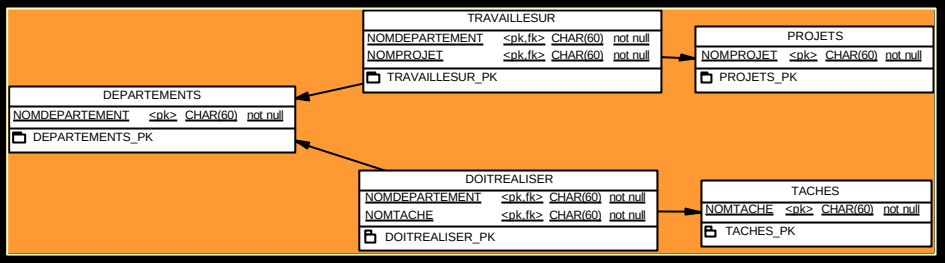
20/07/2000 03:04:44 © Olivier Dahan 30

1er cas : les projets et les tâches peuvent être partagés par plusieurs départements :

On notera les cardinalités 0,N sur l'ensemble des branches.

4 NF - MPD Exemple

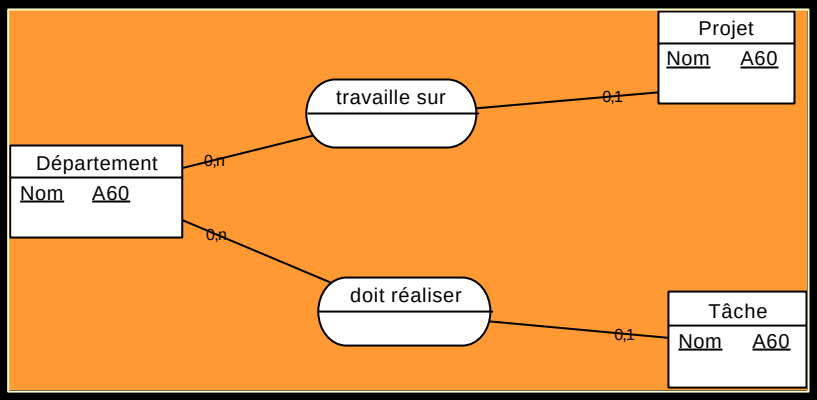
MPD 1er Cas



On note ici l'apparition de deux tables supplémentaires chargées de maintenir les relations 0,N.

4 NF - MCD exemple 2

2ème cas : Les projets et les tâches ne peuvent être affectés qu'à un seul département à la fois



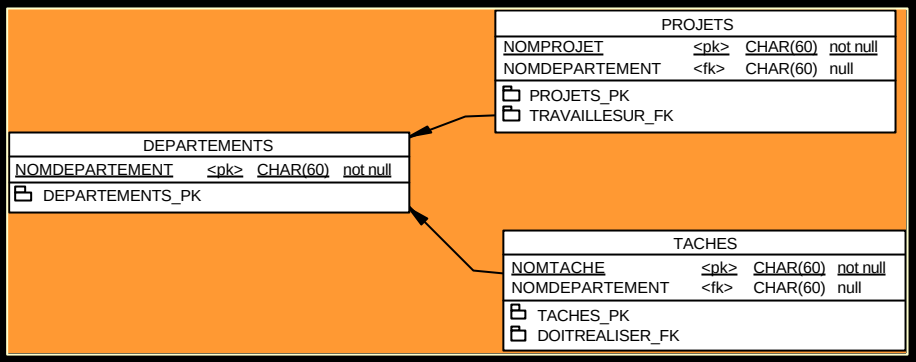
20/07/2000 03:04:44 © Olivier Dahan 32

2ème cas : Les projets et les tâches ne peuvent être affectés qu'à un seul département à la fois

On note ici les cardinalités 0,1 du côté des projets et des tâches.

4 NF - MPD exemple 2

MPD 2ème Cas



Il est clair que partant d'un MCD précisant dès le départ les cardinalités (que seule l'analyse fonctionnelle peut permettre de déterminer) on aboutit à des représentations physiques qui ne ressemblent en rien aux tables de l'exemple.

Ce petit exercice sur un outil CASE nous permet ainsi de mettre en évidence l'utilité d'un formalisme respecté dès le début de la conception d'une base de données.

Accessoirement, on voit l 'utilité de formaliser les relations entre les entités manipulées par un logiciel dès sa conception. Des questions aussi simples que les cardinalités des relations permettent de « toucher » les points sensibles de l 'application.

5 NF - 5ème Forme Normale

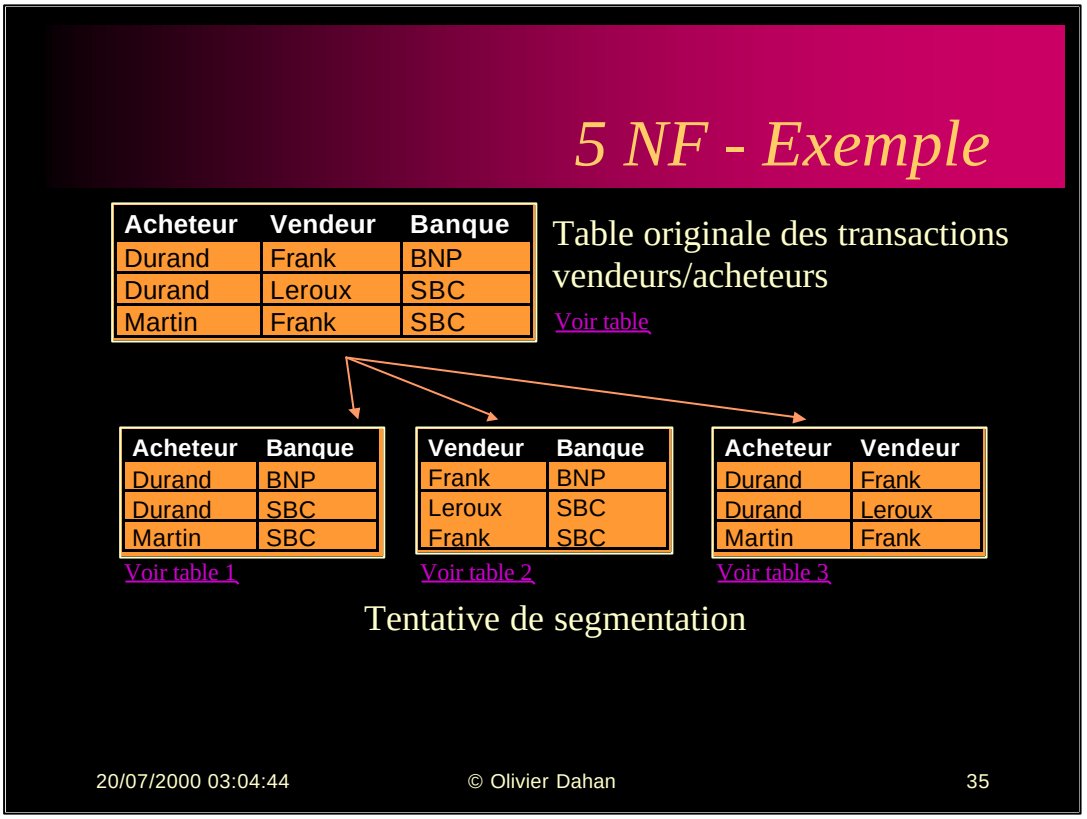
- Aussi appelée “ join-Protection ” ou JPNF ou forme normale Protection-Join. Elle repose sur la nécessité de se prémunir contre la perte d’une jointure ou de pallier une anomalie due à l’absence de join-protection.

  Ce problème se rencontre lorsqu’on a une relation N-vers-N où N > 2. Il existe une méthode de vérification rapide de la 5NF : regarder si la table est en 3NF et si toutes les clés candidates sont des colonnes uniques.

La 5ème forme normale est aussi appelée “ join-Protection ” ou JPNF ou forme normale Protection-Join. Elle repose sur la nécessité de se prémunir contre la perte d’une jointure ou de pallier une anomalie due à l’absence de join-protection.

Ce problème se rencontre lorsqu’on a une relation N-vers-N où N > 2.

Il existe une méthode de vérification rapide de la 5NF : regarder si la table est en 3NF et si toutes les clés candidates sont des colonnes uniques.



Soit une table enregistrant l'acheteur, le vendeur et la banque ci-dessus.

Cette table est une relation triple, mais la majorité des outils CASE n'autorise que des relations binaires. De fait, on pourrait être tenté de reformuler cette table dans un schéma E-R* sous la forme de trois relations binaires qui donneront naissance à trois tables :

E-R: Entité-Relation

5 NF - Requête avec faute de Join-Protection

```
SELECT AV.Acheteur, VB.Vendeur, AB.Banque
      FROM table6c1 as AB,  table6c2 as VB,
table6c3 as AV
WHERE AB.acheteur = AV.Acheteur AND AB.banque =
VB.Banque AND VB.vendeur = AV.vendeur
```

Acheteur	vendeur	Banque
Durand	Frank	BNP
Durand	Frank	SBC
Durand	Leroux	SBC
Martin	Frank	SBC

Table réponse

Anomalie !

Voir requête

Rappel : Table originale

Acheteur	Vendeur	Banque
Durand	Frank	BNP
Durand	Leroux	SBC
Martin	Frank	SBC

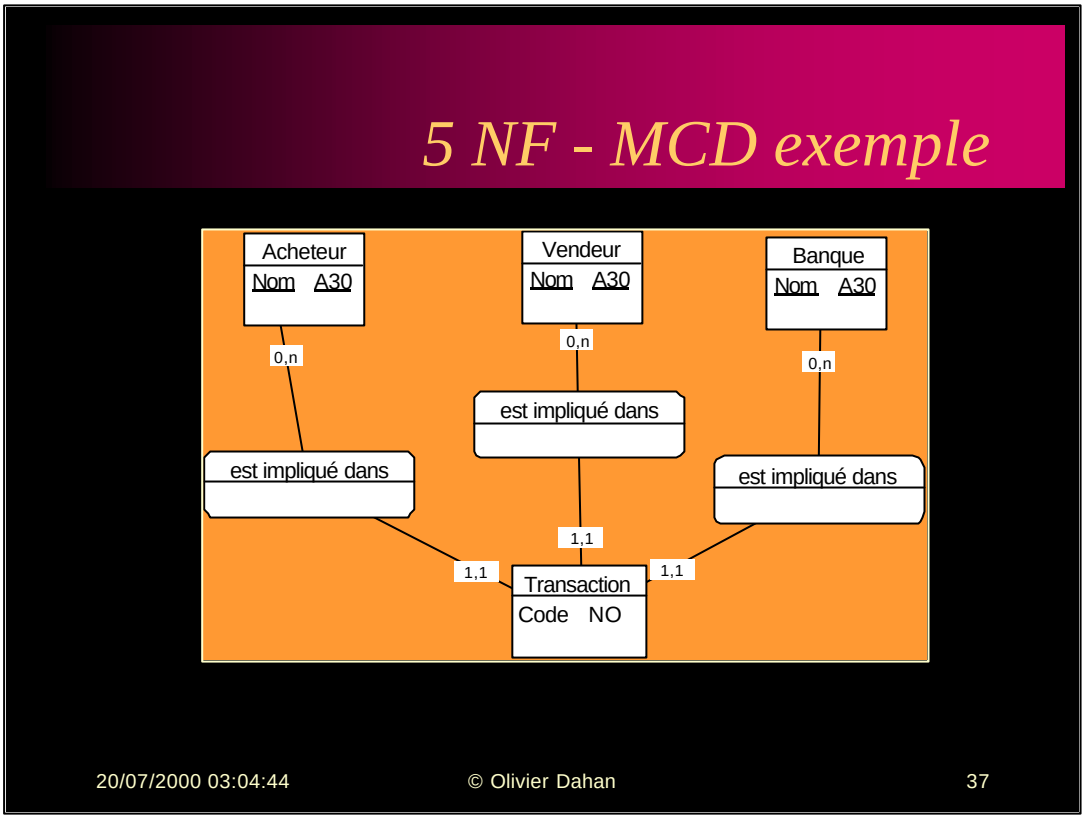
Il y a un problème quand on essaie d'assembler l'information originelle en joignant ces trois tables par paires ce qui sera réglé par l'ordre SQL suivant :

```
SELECT AV.Acheteur, VB.Vendeur, AB.Banque
            FROM table6c1 as AB,  table6c2 as VB,  table6c3 as AV
WHERE AB.acheteur = AV.Acheteur AND AB.banque = VB.Banque AND VB.vendeur =
AV.vendeur
```

Toutefois le résultat de cette requête comptera des lignes comme (Durand, Frank, SBC) qui n'existent absolument pas dans la table d'origine ! C'est ce qu'on appelle une anomalie de join-protection.

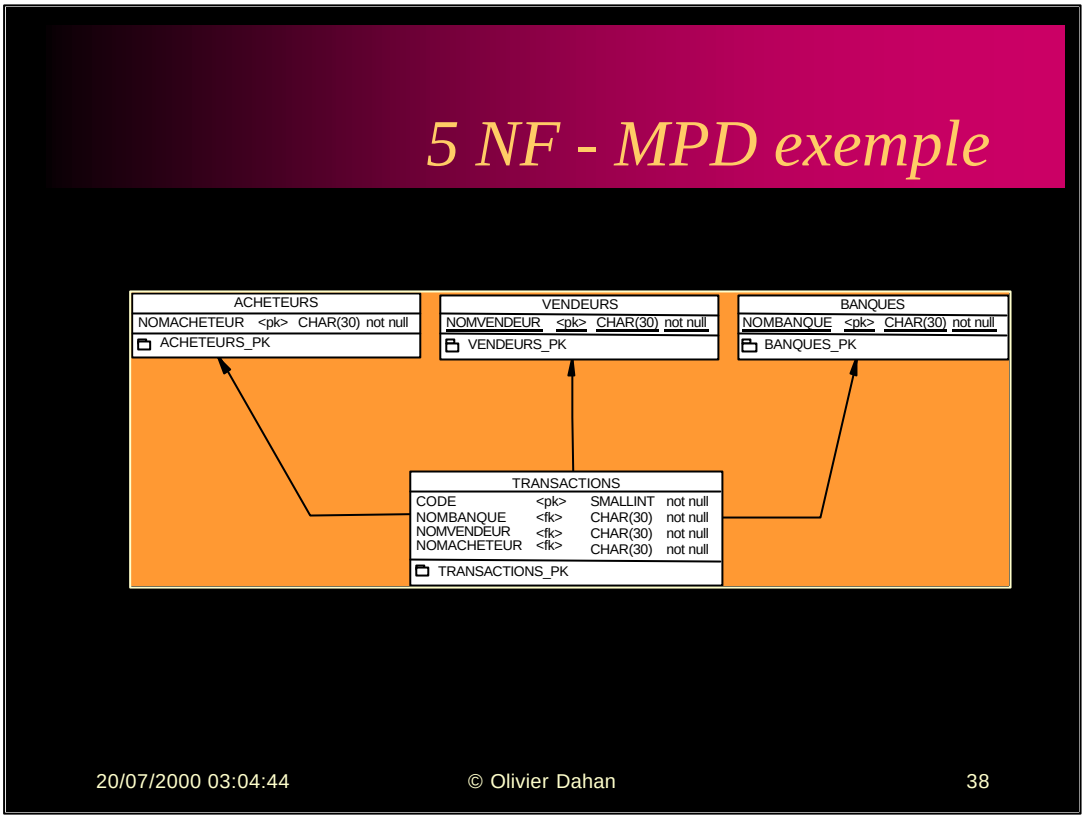
> Il faut d'ailleurs différencier les JPNF puissantes et les JPNF surpuissantes qui font usage de dépendances de jointures (JD). Malheureusement il n'existe aucune manière systématique de trouver un schéma en JPNF ou 4NF parce que le problème est connu pour être NP-complet.

Pour voir la requête en action: cliquer sur « voir requête », passer sur l 'onglet « SQL ». Popup zone sql -> charger mémoires. Puis prendre mémoire 1 et exécuter...



En fait il est possible d'éviter (ou tout cas de minimiser largement) les anomalies de join-protection en concevant de bons modèles conceptuels. Si nous reprenons le même exemple et que nous établissons un MCD correct, nous obtiendrons quelque chose comme l 'image ci-dessus.

On voit qu'ici nous avons isolés trois entités bien différentes l'acheteur, le vendeur et la banque et que l'analyse du problème nous force à créer une quatrième entité qui est la transaction. Une transaction possède un code (ici séquentiel) qui est sa clé primaire, elle entre en relation avec les trois autres entités. Vous noterez les cardinalités, 0,N du côté des entités de base de notre exemple et 1,1 du côté de la transaction. Si nous n'avions pas ajouté une clé primaire à la transaction (" code "), nous aurons utilisé des liens identifiants à la place de simple liens 1,1.



Sous cette forme l'ensemble de tables est en 5NF car nous avons créé une table qui symbolise conceptuellement et physiquement la relation sous-jacente de l'exemple qui ne l'exprimait que de façon virtuelle.

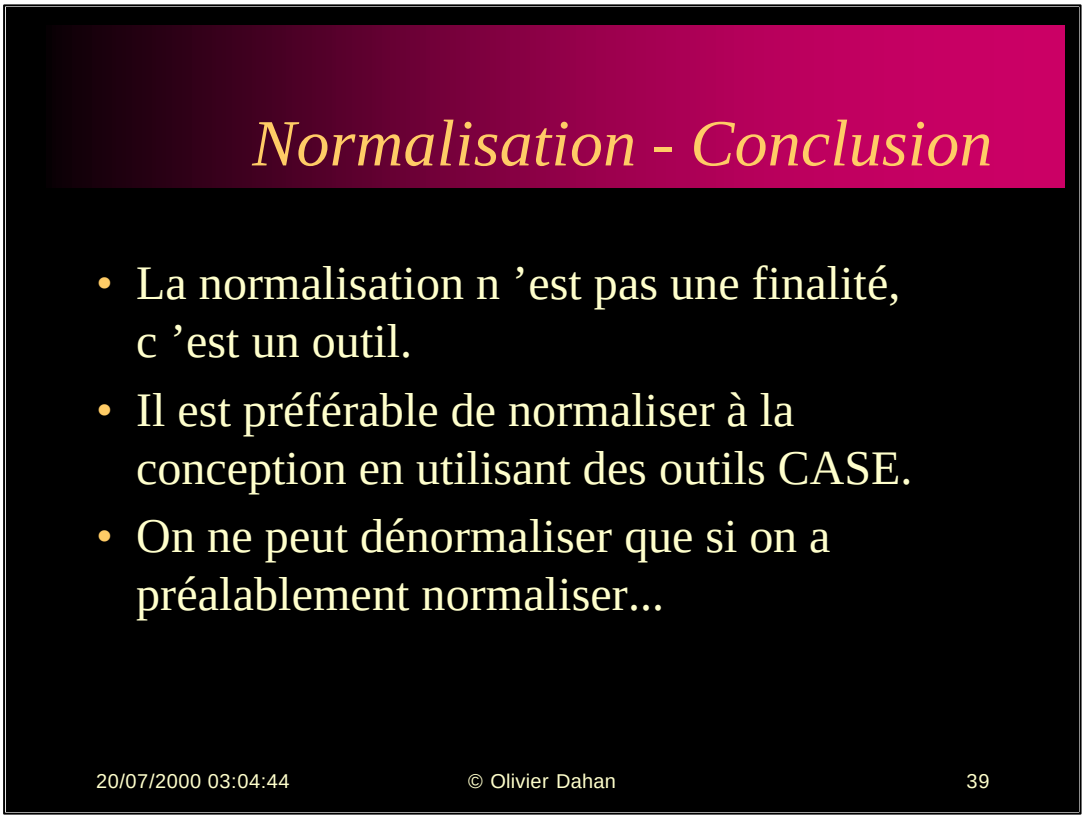
Souvent répondre à la 5NF n'est qu'un problème d'analyse. L'exemple original masquait l'existence d'une entité particulière qui était la transaction en étant lui-même la table des transactions...

Il faut, par contre, toujours gardé présent à l'esprit que la normalisation est un travail qui peut être entrepris à deux instants différents : à la conception de la base de données, ce qui est préférable (nos exemples à l'aide d'outils CASE le montrent assez bien), ou sur une base de données existante. Le plus difficile étant bien entendu de vérifier, voire de corriger, une base de données existante non normalisée.

Forme normale Domain-Key. Nous ne citerons cette forme que pour être complet car sa définition et les méthodes de normalisation afférentes deviennent fort complexes. Cette forme a été décrite et nommée par R. Fagin en 1981

Une dépendance fonctionnelle (DF) a un système défini d'axiomes qui peuvent être utilisés pour la normalisation. Il existe six axiomes connus sous le nom d'axiomes d'Amstrong qui peuvent être appliqués à la résolution de la DKNF. Certains algorithmes comme ceux développés par P.A. Bernstein (1976) permettent de se libérer des problèmes des redondance des DF.

Nous n'entrerons pas dans les détails de cette forme normale et nous renverrons le lecteur à la littérature spécialisée.



Il y a d’un côté les puristes qui considèrent que ne pas amener un schéma au niveau minimum de la 3NF est un crime, de l’autre ceux qui par des ALTER TABLE ajoute et supprime des colonnes dans leurs bases de données sans trop se soucier de l’intégrité de l’ensemble.

Comme souvent la solution se trouve quelque part entre les extrêmes...

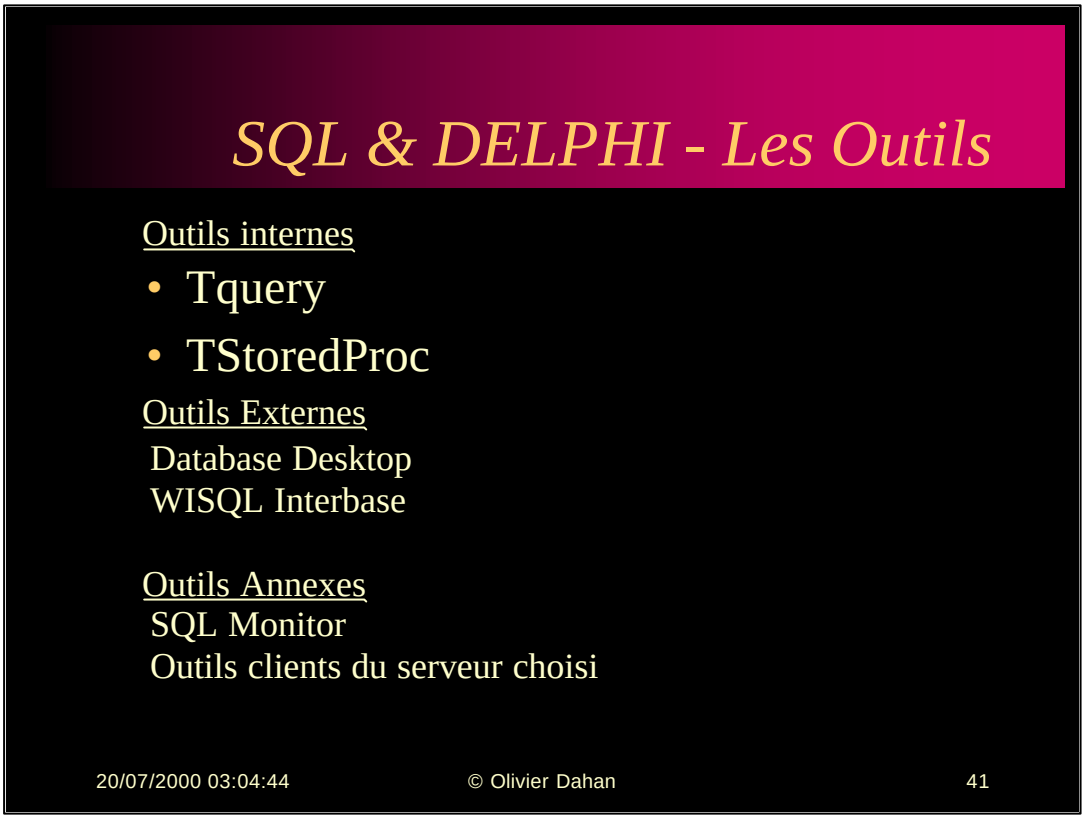
Le bon conseil est certainement celui d’utiliser un outil CASE pour concevoir de bons modèles conceptuels puis de générer les MPD et, ponctuellement, d’appliquer des dénormalisations justifiées dont on assumera, dès le départ, la totalité des conséquences.

Les gens qui sont issus de l’informatique lourde ou qui n’utilisaient pas de SGBD-R ont tous l’habitude de concevoir des tables de grande taille qui contiennent de nombreuses colonnes. Cela était justifié lorsqu’il fallait monter et démonter des bandes magnétiques, cela l’était déjà moins avec l’apparition des disques durs, cela n’est plus raisonnable du tout dès qu’on adopte un SGBD-R. Toutefois la normalisation entraîne la multiplication des tables ne contenant que peu de colonnes. Accéder aux données utilisera de nombreuses jointures ce qui peut ralentir les applications. De même, la saisie peut s’avérer plus complexe à mettre en œuvre avec un schéma normalisé.

A chacun de savoir où s’arrêter dans la normalisation d’une base, la seule chose qui compte réellement est peut-être, simplement, de savoir ce que veut dire normalisation et de ne dénormaliser qu’en parfaite connaissance des causes et effets. Le savoir est toujours préférable à l’ignorance...

Dans cette section nous aborderons des applications pratiques du SQL au travers d'appels directs tels qu'ils peuvent être fait dans le module de base de données de DELPHI ou dans le WISQL d'Interbase ou par le biais d'un TQUERY en programmation sous DELPHI. De fait, nous illustrerons parfaitement la parfaite indépendance entre applicatif et gestion des données (Cf. : Règles 8, 5 et 10 de Codd).

Nous n'aborderons pas directement la programmation des bases de données avec DELPHI, thème qui mérite plus qu'un simple exposé. Rappelons que la VCL est dotée de deux palettes de composants entièrement dédiés à la gestion des données et que la nouvelle architecture multi-tiers depuis DELPHI 3 ouvre de nouveaux horizons (sources de données non BDE, MIDAS, …) tout en bénéficiant d'une même interface de développement.



Côté SQL, les outils principaux mis à disposition par DELPHI sont le TQUERY, qui permet l'exécution de requêtes, et le TStoredProc qui donne accès aux procédures stockées dans le serveur. D'autres composants jouent un rôle essentiel, comme le TDataBase (assurant la liaison avec une base ou permettant de gérer les transactions) ou le Tsession (contrôlant les sessions ouvertes).

Hors des applications, DELPHI fournit des outils comme le Database Desktop (logiciel Paradox amputé des impressions et de la gestion des fiches) ou comme WISQL fourni avec Interbase.

Dans l 'IDE, DELPH fournit aussi un SQL Monitor permettant de tracer la conversation entre l 'application et le serveur.

Enfin, la base cible choisie (Oracle, Sybase…) offre souvent des outils spécifiques de maintien des bases.

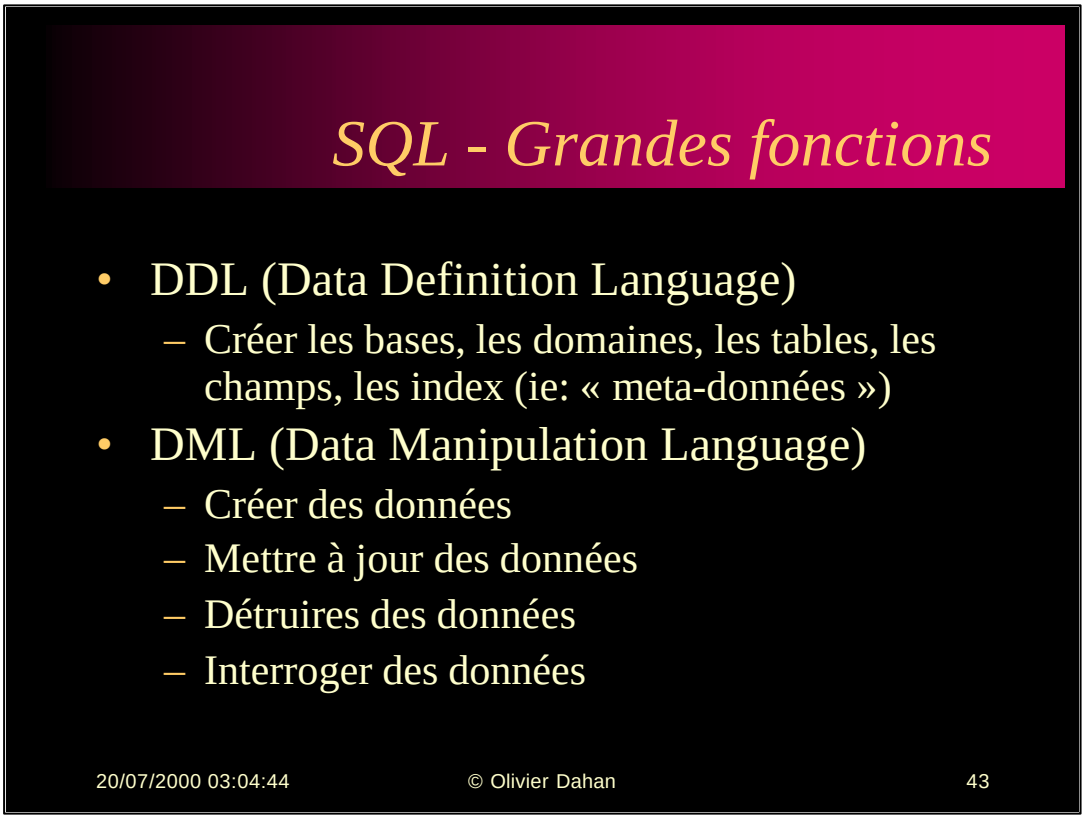
Il existe actuellement plusieurs normes SQL ANSI / ISO. Le standard SQL-89 est celui qui est le mieux supporté aujourd'hui par l'ensemble des serveurs du marché. SQL-92, qui est compatible avec SQL-89, ajoute de nombreuses possibilités mais son support est loin d'être égal. Il existe des normes encore plus récentes mais pour lesquelles personne ne s'attend à voir de réel support avant longtemps (même si, ponctuellement, les versions récentes de certains serveurs comme Interbase proposent déjà des implémentations de certaines fonctionnalités).

Il est ainsi très difficile de parler du SQL de façon générique puisqu'il existe autant de version de ce langage qu'il y a de SGBD-R sur le marché. Nous utiliserons ici le SQL local du BDE 5 ou celui d'Interbase. Il est intéressant de noter que Interbase de Borland offre un support de bonne qualité du SQL-92 notamment syntaxique là où d'autres serveurs pourtant de réputations mondiales comme Oracle ou Sybase n'offre parfois que des syntaxes exotiques non conforment aux normes.

La norme SQL-92 se présente sur trois niveaux : restreint, intermédiaire et complet. Même si la tendance va vers un support complet, on aura compris qu'aucun produit du marché n'offre pour l'instant ce niveau.

Pour terminer avec les généralités, ajoutons que le NIST (National Institute for Standards and Technology) aux USA propose un ensemble de FIPS (Federal Information Processing Standards) qui comprennent des séquences de test pour différents langages de programmation, dont SQL (séries FIPS-127). D'ailleurs, pour faire une offre de contrat au gouvernement américain, il faut avoir passé avec succès les tests des FIPS. Les FIPS concernant SQL sont segmentés en plusieurs couches répondant à des fonctionnalités précises de telle sorte que les éditeurs puissent se faire certifier pour un niveau donné sans être obligés de supporter tout SQL-92 ou être rejeté définitivement.

Enfin, on pourra s'intéresser aux travaux du SQL / Access Group qui est un consortium d'éditeurs traitant de l'interactivité entre bases de données ou à ceux de X / Open, autre consortium d'éditeurs traitant de la portabilité du langage SQL. La plupart de éditeurs appartenant à ces groupes sont membres du comité de standardisation X3H2 ANSI s'intéressant au SQL.



SQL se présente comme un langage à deux couches : le DDL et le DML.

Le DDL (Data Definition Language) permet de créer des bases de données, les tables qui en font partie, les champs, les index, etc.

Le DML (Data Manipulation Language) permet d'exploiter les données : ajouter, supprimer des enregistrements, interroger les données.

A côté de ces couches dédiées on trouvent le plus souvent un langage procédural dérivé de SQL qui permet de créer des procédures stockées ou des triggers.

Les trois fonctions principales du DDL sont : CREATE TABLE, DROP TABLE et ALTER TABLE.

Des fonctions annexes sont aussi proposées pour manipuler de façon isolée les contraintes et les index.

On trouve aussi des fonctions dédiées à la manipulations des méta-données comme les domaines.

Certains serveurs (comme Interbase) proposent aussi la création de bases de données directement en SQL.

Les grandes fonctions du DML

- INSERT, création d ’enregistrements
- DELETE, suppression d ’enregistrements
- UPDATE, mise à jour d ’enregistrements
- SELECT, interrogation des données

Les principales fonctions sont ici : INSERT INTO, DELETE FROM, UPDATE et SELECT

Ces ordres permettent, respectivement, d’insérer de nouveaux enregistrements, de supprimer, ou de modifier des enregistrements existants et enfin d’interroger la base de données.

Voici la syntaxe Interbase des trois premiers, l’ordre SELECT étant présenté plus détail à la section suivante :

```
INSERT INTO <object> [(col [, col ...])]
{VALUES (<val> [, <val> ...]) |
<select_expr>};

DELETE FROM TABLE [WHERE <search_condition>];

UPDATE {table | view}
SET col = <val> [, col = <val> ...]
[WHERE <search_condition>;
```

L ’ordre SELECT

Syntaxe du BDE local:

```
SELECT [DISTINCT] liste_colonnes
      FROM référence_table
      [WHERE condition_recherche]
      [ORDER BY liste_tri]
      [GROUP BY liste_groupe]
      [HAVING condition_having]
      [UNION expr_select]
```


Il permet de ... sélectionner des données dans une base de données relationnelle. Il est capable de comprendre les jointures et de croiser les différentes données pour fournir le résultat.

L’importance de cette instruction qui illustre parfaitement l’aspect relationnel fait qu’il bénéficie de sa propre section que voici.

SQL - Démonstration

Pour illustrer les divers aspects de SELECT et la mise en œuvre des jointures nous utiliserons le MK Query Builder.

Exécuter MKQB

MK QUERY BUILDER fait l ’objet d ’une présentation complète.
Voir le programme des I-CON 98 pour connaître l ’heure de ces sessions.

Après avoir abordé les fondements conceptuels des SGBD-R (règles de Codd), la normalisation des bases de données ainsi que les grandes lignes du SQL l'auteur espère non pas vous avoir converti au relationnel ni même vous avoir enseigné l'art de la normalisation, mais vous avoir simplement donné l'envie à la fois de lire la littérature spécialisée et celle de mettre en œuvre des applications bien conçues à l'aide de DELPHI qui est, aujourd'hui, le seul vrai outil RAD compilé orienté objet avec C++ Builder (moins pratique toutefois en raison de la structure même du langage et des restrictions imposées par la norme ANSI).

Au-delà de cet aspect technique (toutefois essentiel) DELPHI est un environnement qui donne un grand plaisir toujours renouvelé, et prendre du plaisir à développer sous Windows dont l'opacité des API est plus que légendaire, si ce n'est pas un miracle, cela y ressemble beaucoup...

Et L'avenir ?

Une fois que vous serez parfaitement à l'aise avec la programmation orientée objet et avec les SGBD-R, vous vous rendrez vite compte que la première créée de nouveaux besoins auxquels les secondes ne peuvent pas totalement répondre. Le Relationnel a ses limites. Demain, les bases de données objet seront une réalité en micro-informatique. On trouve d'ors et déjà certaines bases de ce type à des prix attractifs, toutefois elles restent assez lentes et peu ou pas intégrées dans les grands EDI du marché. Mais cela viendra bientôt...

D'ailleurs, lorsque je concluais ainsi l'année dernière je ne savais pas encore que les mois qui suivraient me donneraient tant raison… En effet, les extensions Objet de Oracle 8, même si elles ne sont qu'un début partiel et timide, démontrent que l'avenir est bien à l'objet, même pour les bases de données…

Raison de plus pour vous intéresser dès maintenant à la programmation orientée objet ainsi qu'aux concepts des SGBD-R... Il est toujours plus difficile de prendre un train en route que d'être déjà installé quand il démarre...

.