BD40:

INITIATION AUX BASES DE DONNEES

P.Deschizeaux

Chapitre 1:
Introduction aux bases de données.

1.1. GENERALITES

L'information sous toutes ses formes (données numériques, textes, image, son) tient une place de plus en plus grande dans le monde moderne. Tous les secteurs de l'activité humaine sont touchés et se trouvent confrontés à des problèmes de plus en plus complexes.

1.1.1. La nature des information est de plus en plus complexe: on est passé d'informations purement textuelles en 1980 à des données images et son, voire à des films.

1.1.2. La taille des informations traitées augmente (avec la taille des disques support)
exemple: fichier de la sécurité sociale: 50M de français, admettons 10000 caractères par personne (1 page environ)➔500 giga octets, c'est faisable; Mais imaginons une loi imposant la photo de chacun: une photo= 1 Moctet ➔50 000 giga octets! ! !

1.1.3. La dimension physique des systèmes s'accroit : en 1960 un seul site; en 1980 plusieurs sites dans la même ville, en 2000 sur plusieurs continents.
Exemple:Trains SNCF: il y a 1000 gares en France, donc potentiellement 1000 personnes au moins qui peuvent demander simultanément des informations sur les trains. C'est infaisable sur un site unique, même via INTERNET: on est obligé de décentraliser l'information: une base dans chaque gare (et alors problèmes de cohérence des informations!)

1.1.4. La diversification des informations s'accroît:
exemple: fichier des pièces détachées de Renault: 10000 pièces par modèle, environ 100 modèles soit 1M descriptions. Pour chaque pièce on a des informations très diverses: cotes, matériaux, informations sur le fournisseur, schéma de montage, photo , etc...

deux hypothèses:

ou bien on a un format de données standard (mais c'est idiot: on n'a pas besoin de photo d'un boulon!)

ou bien on a un format variable au prix d'une complexité de gestion considérable.

1.1.5. Parallèlement, la diversité des recherches d'informations dans la base s'accroît :
Il y a 20 ans, consulter les données revenait à imprimer un "listing" , en général énorme, et à dépouiller à la main. Actuellement, vu la taille des systèmes d'informations, ceci serait impossible. On dispose de "requêtes" permettant d'extraire des informations triées du système; exemple: sortir tous les clients d'une grande surface ayant pour plus de 1000F d'achat par an. Il est clair qu'on ne peut imprimer tous les clients (avec leurs achats) puis trier à la main!

1.1.6. Les contraintes deviennent plus sévères:

contraintes de temps:
La transmission d'images animées et surtout du son impose des contraintes de synchronisation très sévères.

Contraintes de sécurité: l'usage d'informations confidentielles impose des dispositifs anti-intrusions sophistiqués.

Contraintes légales: loi "informatique et liberté" on n'a pas le droit de mettre n'importe quoi dans une base de données (informations sur des personnes, à caractère raciste, religieux, médical, etc...). Les informations sur des personnes doivent être déclarées à la CNIL.

Conclusion:
Dans tous les cas on se trouve confronté à des problèmes de performances. Ces performances vont être très liées à l'organisation des données.

Ce qui précède montre qu'un système d'information (on dit aussi "base de données") est quelque chose de très complexe:

Il y a 20 ans, les données étaient rangées dans des "fichiers" indépendants, dont la structure et l'exploitation étaient définis dans un "langage de programmation universel".
Par exemple en PASCAL on aurait défini un fichier des clients d'une entreprise en définissant la nature et le format des données.
Exemple

```
Type client= record
                    nom: string[50];
                    numero:integer;
                    rue: string[50];
                    code_postal:integer;
                    ville:string[50];
              end;
var fichier_client: file of client;
```

La recherche d'information (par exemple la recherche des clients parisiens) demandait que soit écrit un programme spécifique en PASCAL; L'inconvénient majeur était que cette technique nécessitait des gens compétents.

Actuellement, ne base de donnée ne se réduit pas à un ensemble de fichiers indépendants: on utilise un "SGBD"= Système de Gestion de Bases de Données, qui comporte:

Un outil de définition du format des données.
Un outil ergonomique de saisie des données,
Des outils d'interrogation de la base (requêtes) utilisables par des non spécialistes,
Des outils de vérification de la cohérence des données,
Des outils de compression des données,
Des outils de gestion de la base (édition, copie, mise à jour, effacements, etc..)
etc.

NOTA: Il existe de nombreux SGBD, l'outil utilisé en BD40 est ACCESS, logiciel fourni par Microsoft pour PC, actuellement très répandu.

Le cours sera axé sur deux points

La conception optimale de base de données. Il s'agira de faire des choix d'organisation en fonction des besoins et des contraintes de place et de durée d'exécution. On utilisera la méthode MEURISE comme support de réflexion , des compléments sur d'autre méthodes (NIAM, base de données hiérarchiques) seront plus brièvement étudiées. Cette méthode sera appliquée en ACCESS sous forme de TP.

La réalisation d'une base de donnée sera étudiée en second lieu: étude de la représentation physique des données, étude des algorithmes de traitement d'information.

1.2. Conception de systèmes d'information.

Le problème est donc de définir l'organisation interne de la base de données. Cette organisation résultera d'un compromis entre les besoins de utilisateurs de la base (essentiellement quelles seront les informations qui en seront extraites?) et les contraintes techniques (vitesses de traitement et taille mémoire principalement).

Les contraintes de temps sont extrêmement sévères; malgré des performances en croissance continue des ordinateurs, on arrivera parfois à des temps de calculs prohibitifs si les données sont mal organisées. On montre ci dessous par exemple rechercher une facture particulière dans un fichier d'un million de factures peut coûter 20 opérations ou 1 million d'opérations suivant que les données sont bien ou mal organisées

Les contraintes de place en mémoire seront souvent moins sévères, mais associées à des contraintes de cohérence des données, conduiront toujours à éviter la duplication d'information, et donc à choisir où les informations doivent se situer, et comment y accéder.

1.2.1. Le problème de durée de recherche d'informations.

Les données sont rangées dans des fichiers sur disque. Ces fichiers subissent des opérations telles que:

création,
mise à jour,
destruction,
consultations,
etc.

Comme ces fichiers peuvent être de taille importante, le coût (en temps) de ces opérations peut ne pas être négligeable . Examinons quelques exemples:

exemple 1: fichier du personnel d'une entreprise = suite de N enregistrements de la forme {nom, prénom, adresse, date de naissance, etc..}. Ces enregistrement ont été entrés dans le désordre au fur et à mesure des recrutements.

Posons nous la question du coût d'une consultation par exemple pour rechercher l'adresse d'une seule personne X.

Le programme sera approximativement:

```
lire le fichier et le ranger dans un tableau d'enregistrements T
i=0;
répéter
        i=i+1
jusqu'à  T[i].nom= X;
ecrire T[i].adresse;
```

Statistiquement on devra répéter N/2 fois l'analyse. Si on cherche les adresses de toutes les personnes, cela coûtera $N^2/2$. (voir également "algorithmes de recherche" au chapitre 7)

Exemple2: considérons le cas du même fichier trié par ordre alphabétique des noms.

Le programme sera le suivant:

```
lire le fichier et le ranger dans un tableau d'enregistrement T;
a=0;b=n;
répéter
        c=(a+b)/2;
        si T[c].nom>"X" alors b=c    {le client est dans la première moitié du tableau}
                       sinon a=c     {le client est dans la deuxième moitié du tableau}
jusqu'à  T[c].nom= "X";
écrire T[c].adresse;
```

Quel en est le coût?

Exemple N=1024; on cherche successivement dans des zones de tailles 1024, 512, 256, 128,64,... 2,1. Soit 10 comparaisons (au lieu de 1024 si le fichier n'est pas trié). Plus généralement le coût est $\text{Log}_2 N$. On voit qu'on a intérêt à trier le fichier!

Exemple 3: supposons cette fois que les employés sont numérotés; on pourra chercher l'adresse de l'employé N° i en une seule opération: *écrire t[i].adresse*

On constate qu'il est préférable de faire la recherche sur des numéros! Néanmoins, il est clair que savoir le numéro de la personne recherchée sera parfois difficile.

REMARQUE: corrélation taille/temps de calcul.

a) Les systèmes d'exploitation rudimentaires (par exemple WINDOWS) perdent beaucoup de temps dans les échanges disque <==> unité centrale. On en déduit que si un gros fichier doit être chargé intégralement en mémoire pour être traité, le temps cumulé (traitement + chargement) peut devenir prohibitif.

On aura donc toujours intérêt à fractionner l'information ("diviser pour règner"):

- fractionner **physiquement**: mieux vaut par exemple avoir un fichier de client par année qu'un seul regroupant toutes les années. Les années antérieures sont sur CDROM, l'année en cours sur le disque dur.

-fractionner **géographiquement:** par exemple, on ne fait pas un fichier de tous les employés de l'éducation nationale , mais on a un fichier par académie.

- fractionner **logiquement,** séparer les informations de nature différente; par exemple, bien que hommes et machines puissent être considérés comme des "ressources" dans une entreprise, on sépare en deux fichiers.

b) **la décentralisation géographique** accentue encore le résultat: accéder à des données ne revient pas seulement à les charger en mémoire; il faut en plus compter les temps de transfert à distance (via le réseau).

Exemple de problème: savoir si l'étudiant X est admis dans une école d'ingénieur. La requête comporte plusieurs accès à des données

- quelle est la décision du jure d'admission.
- a t'on confirmation de sa nationalité (il est né à Tahiti)
- a t'on confirmation de sa mention au bac (passé à Kourou)

Il est clair que si, pour exécuter le traitement, il faut se faire envoyer tout le fichier d'état civil de Tahiti, tout le fichier du bac depuis Kourou, pour vérifier soi même, ça prendra du temps! La bonne solution est d'éclater la requête: demander seulement une confirmation à Tahiti et à Kourou . on parlera de requête décentralisée.

2.2. Problèmes de place mémoire: Insuffisances du principe "tableau"

exemple: considérons un fichier d'état civil: chaque enregistrement comporte les caractéristiques d'une personne sous un format fixe:

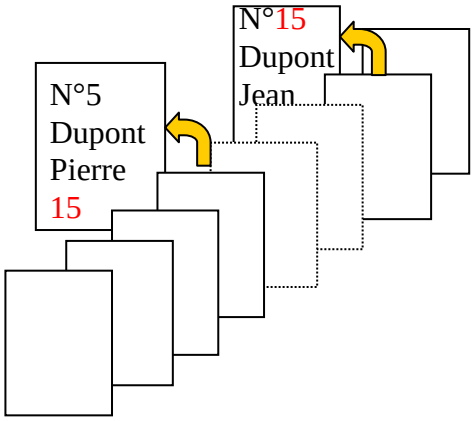
nom, prénom, date et lieu de naissance
identité du père et de la mère,
prénoms et dates de naissance des enfants

Ce fichier présente deux inconvénients majeurs (CF entités):

a) ce fichier est redondant: l'identité des parents fait double emploi avec leurs propre fiche d'état-civil; il est clair qu'on a un fichier 3 fois trop gros.

Comment pallier à ce défaut?

Reprenons la vielle technique des fiches cartonnées rangées dans un tiroir. Ces fiches sont numérotées; donc pour obtenir l'identité du père de quelqu'un, il n'est pas nécessaire de l'avoir inscrite sur la fiche, on se borne à indiquer le numéro de la fiche du père



Le système a évidement deux inconvénients :

- on a crée une information supplémentaire, le numéro de fiche,

- pour obtenir les informations sur le père, il faut chercher dans tout le tiroir. Mais cet inconvénient est mineur tant que les fiches sont **triées**

En informatique, on va copier cette technique: imaginons un tableau (style EXCEL) tel que:

N°	NOM	PRENOM	N° du père	adresse	etc...	
5	Dupont	pierre	15	paris	.....	
15	Dupont	jean	37	Belfort	.....	

Pour chercher les informations sur le père, il suffit de chercher la ligne dont le numéro est indiqué dans la ligne du fils. C'est évidement plus facile par informatique qu'à la main.

Définition: l'information "numéro d'une ligne du tableau" est appelée un pointeur..

Remarque:
La recherche de la ligne indiquée peut prendre du temps; cela dépend de **comment sont rangées les données.**

- si les lignes sont numérotées dans le désordre, il faut les parcourir toutes (plus exactement la moitié en moyenne) pour trouver la bonne

- si les lignes sont rangées dans l'ordre des numéros c'est évidement beaucoup plus facile, surtout si tous les numéros sont présent

- un problème se présentera toutefois si on entreprend de supprimer des lignes.

Deuxième inconvénient: Le nombre d'enfants est variable:

On est obligé de prévoir des emplacements pour les cas extrêmes (disons 20 enfants maximum!). Mais alors dans 99% des cas ces emplacements sont vides, les familles françaises ayant en moyenne 1.5 enfants.

Exemple

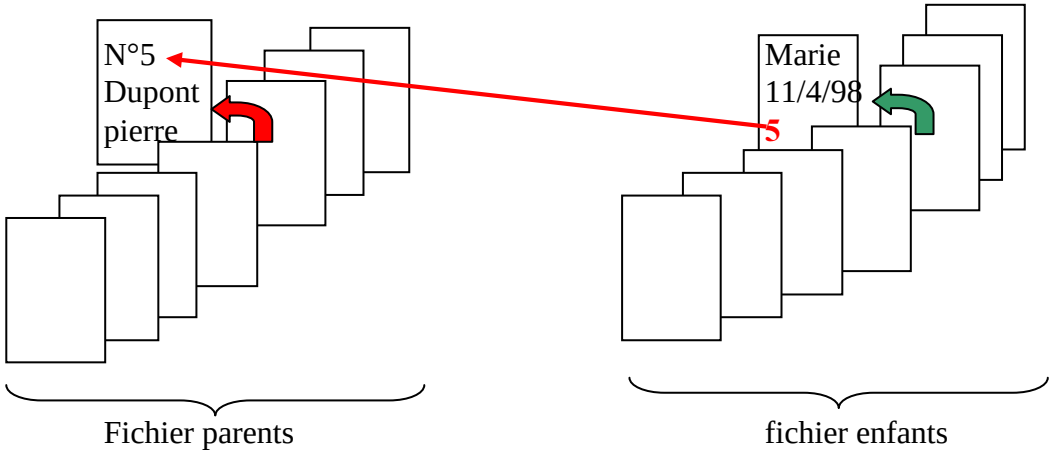
NOM	PRENOM	enfant 1	enfant2	enfant3	enfant 4	.....	.....	enfant 20
Dupont	jean	pierre	marie	sophie				

Dupont	pierre	jacques						
Durand	Alain							
Martin	bernard	julie						

On a énormément de place perdue; ce serait encore plus marquant si les informations sur les enfants étaient plus longues (exemple prénom, date de naissance, cursus scolaire, etc...). Il est clair qu'on ne peut procéder ainsi pour des fichiers comportant des centaines de milliers de personnes.

On pourrait évidement essayer de tourner la difficulté en prévoyant un nombre plus restreint d'enfants (par exemple 10) mais que faire des familles nombreuses (les mettre dans un fichier spécial? mais alors que faire quand le nombre d'enfant croît ou décroît)?

Solution:
On va **éclater** l'information en deux catégories: la catégorie des parents, la catégorie des enfants (comme si on avait deux tiroirs de fiches: un pour les fiches parents, un pour les fiches enfants)



Remarques

a) le pointeur est ici dans la fiche "enfant"; il désigne un père unique. (Il y aura évidement dans la même fiche un pointeur vers la mère).
L'inconvénient est évident: rechercher les enfants de quelqu'un nécessitera de chercher dans le fichier "enfants" quels sont ceux qui ont le même numéro que le père. Ce sera évidement plus rapide si les enfants sont triés par numéro de père (mais il ne seront jamais triés à la fois par numéro de père et numéro de mère).

On perd ici en temps de traitement, mais on gagne considérablement en place.

b) Ici nous avons deux fichiers celui des enfants et celui des parents, car les informations les concernant ne sont pas forcément les mêmes (par exemple l'information adresse des enfants est absente, elle est dans le fichier parent)

c) ce formalisme introduit une contrainte de cohérence des données: le numéro de père apparaissant dans la fiche enfant doit évidement correspondre à une personne existante. Des vérifications seront nécessaires:

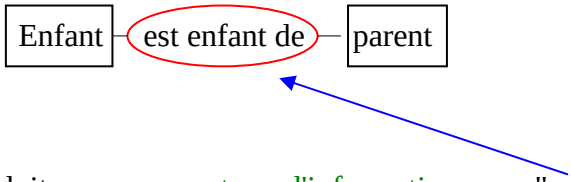
- lors de la saisie d'informations, il faudra vérifier que le père existe; si ce n'est pas le cas, il faut d'abord saisir l'identité du père, puis celle de ses enfants.

- on s'interdira de supprimer une personne du fichier parents sans avoir au préalable supprimé ses enfants.

Ceci forme ce qu'on appelle la contrainte d'intégrité référentielle

Formalisation:

On a introduit une information supplémentaire: la relation de parenté. ceci se modélise par le schéma suivant:



Ceci signifie qu'on a introduit un nouveau type d'information: une "association" dont le rôle est de définir les objets liés entre eux.

Nota: diverses méthodes sont utilisées pour modéliser ceci.

le modèle utilisé ici sera le modèle MERISE (le plus utilisé en France)

dans les pays Anglo-saxons, on utilise le modèle NIAM (décrit en annexe)

Il existe d'autres modèles tels que OMT, UML très utilisés par exemple en génie logiciel. Se reporter aux ouvrages spécialisés

CHAPITRE 2:
Le modèle MERISE

Ce modèle distingue fondamentalement deux niveaux de modélisation:

Le niveau du modèle conceptuel de données (MCD) qui s'intéresse à l'organisation des données en "entités" élémentaires et aux relations entre elles; il est indépendant des langages et outils informatiques utilisés.

Le niveau du modèle logique de données (MLD) qui s'intéresse à la représentation des données définies par le MCD. Ce niveau dépend du type d'outil informatique utilisé.

2.1. Le MCD

Dans le chapitre précédent, on a vu apparaître deux types de d'informations:

- des informations qu'on peut qualifier de principales (par exemple l'état civil d'une personne, une adresse, etc…
- Des informations définissant les relations qui existent entre ces informations principales

Clairement, ces informations ne sont pas de même nature. On convient donc distinguer deux types d'objets:

2.1.1. les entités ou informations principales.

Ces entités contiennent des "occurrences" ou informations particulières décomposées en "champs" appelés aussi "attributs" ou "propriétés".

Par exemple l'occurrence {Dupont, jean, 11 mai 1985, paris} se décompose en 4 champs

Nom = Dupont
Prénom = Jean
Date = 11 mai 1985
Ville = Paris

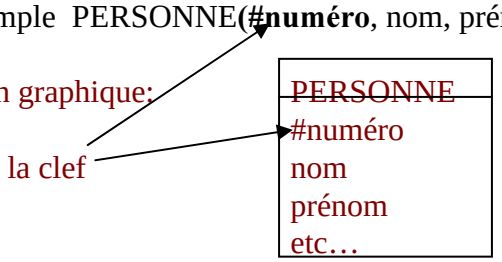
2.1.1.1. Règle: Toute entité devra avoir un "identifiant" appelé aussi "clef" destiné à **repérer de manière unique** les occurrences. Par exemple pour se protéger des

homonymie, on mettra dans l'entité "personne" un numéro (numéro INSEE par exemple).

Cette clef sera représentée par le symbole # précédent le nom de la clef: exemple: #numéro, ou en soulignant, exemple numéro
La clef ne sera pas forcément un numéro; par exemple une voiture sera repérée par son immatriculation, un produit par sa désignation, etc. mais la numérotation sera le moyen le plus "économique".

Notation: exemple PERSONNE(**#numéro**, nom, prénom, adresse, age, . . .)

Représentation graphique:



2.1.1.2.Remarque
On peut trouver des **clef composées**; par exemple une personne sera désignée par le couple [nom, prénom]. Cette situation est à distinguer de cas ou l'entité aurait **plusieurs clefs**, par exemple un produit désigné par son nom (exemple "imprimante CANON BJC2000" **ou** son n° d'inventaire exemple "XB2754A"). On verra plus loin que ceci est absolument à éviter.

2.1.1.3. MLD: une entité est réalisée par un tableau (en ACCESS on dit "table") de valeurs dont chaque ligne est une occurrence, chaque colonne un champ. Ce tableau contient des informations (éventuellement "nulles"). Exemple:

Personne:

#N°	Nom	Prénom	adresse	……
1	Martin	marie		
2	Renault	mégane	Bordeaux	
3	Dupond		Lille	

Le format des données (type et taille des champs) est alors déterminé.

nota: il est très important de définir correctement le format des données pour optimiser l'encombrement mémoire ET les temps de traitement

exemple 1: inutile de définir le champ "salaire mensuel" (en euros) d'un enseignant comme un entier long: un entier cours suffit (hélas!). mais s'il s'agit du salaire des dirigeants d'une multinationale, ce pourra être indispensable.

exemple 2: inutile de définir le champ "nom" d'une personne sous forme d'une chaîne de 250 caractères, c'est de la place perdue!

exemple 3: le champ "téléphone" est une chaîne de caractères et non un entier, car on veut pouvoir écrire 06 35 12 54 87, avec des blancs, il faut également plus de 10 caractères pour les cas de n° à l'étranger.

2.1.1.4. Identifiant relatif

On a vu que toute entité doit avoir un identifiant.; par exemple une personne est identifiée par son numéro. Mais il se peut que cet identifiant soit un numéro **"relatif";** par exemple un enfant sera repéré par

a) le numéro de son père

b) son numéro d'ordre dans la famille

par exemple on dira que "Bernard est le deuxième enfant de la personne n°235"

Voir également au paragraphe "cardinalités."

2.1.2. les associations reliant les occurrences d'entités.

- Ces occurrences reliées peuvent être de nature différente ; par exemple l'association "habite" relie une personne (repérée par son nom) à un domicile (repéré par son adresse)

- Elles peuvent relier des occurrence de même type (par exemple l'association *"est parent de"* relie deux personnes.

Ceci généralise la notion de pointeur vue au chapitre 1. En effet, le problème est que dans la notion de "est père de", chaque enfant a **un seul** père (donc on peut mettre dans l'entité enfant un pointeur vers le père), on va trouver des associations reliant **plusieurs** objets dans les deux sens:

exemple: l'association "habite" reliant une personne et un logement

une personne peut habiter plusieurs logements et un logement peut loger plusieurs personnes. On ne peut donc pas se passer de l'association:

- ni mettre un pointeur vers le logement dans la table personne (certains riches auront 10 logements et les pauvres un seul!) il y aurait de la place perdue pour les pointeurs non utilisés.

- ni mettre un pointeur vers la personne dans l'entité logement: même raison certains logements (chambre d'étudiant) ne logeant qu'une personne, et d'autre (foyer) en logeant des centaines.

Nota: le MCD ne précise pas comment se matérialise l'association, ceci est du ressort du MLD; mais on peut en donner une idée: en général l'association sera formé de deux "identifiants" appelés aussi "clef externe". Le cas le plus commun est celui où les clefs externes sont des numéros. Mais on peut tout à fait imaginer que la clef soit une chaîne de caractères , voire plusieurs (nom, prénom par exemple) .

2.1.2.1 Champs propres à l'association

Il peut se produire que l'association comporte des informations qui lui sont propres

Exemple: l'association "achète" reliant une personne et une voiture comporte une "date d'achat". En effet cette information ne peut être mise dans l'entité "personne" (un personne peut acheter des voitures à des dates diverses), ni dans l'entité "voiture" (une voiture peut être achetée plusieurs fois).

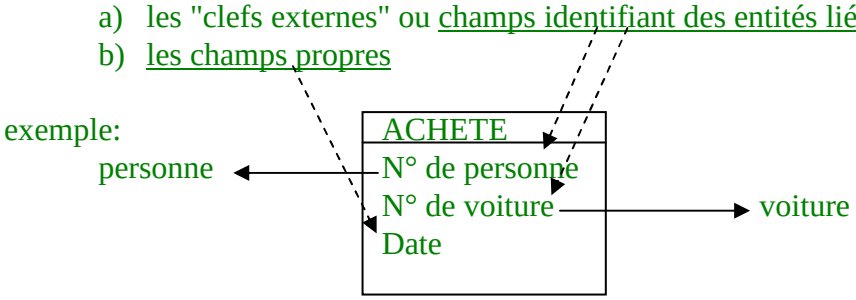
2.1.2.2. Représentation graphique

Nota: Les clefs externes n'apparaissent pas dans le MCD, elles relèvent du MLD.

2.1.2.3. MLD:

La réalisation de l'association se fait à l'aide d'une table comportant:

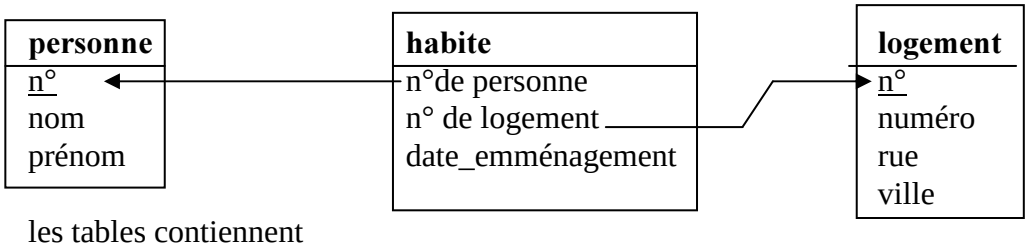
a) les "clefs externes" ou champs identifiant des entités liées
b) les champs propres



2.1.2.4. Collection d'une association:

C'est l'ensemble des informations auxquelles elle donne accès directement; exemple;

on a onc un MLD tel que:



les tables contiennent

personne		
1	Martin	jean
2	Dupont	Sophie
3.....		

habite		
1	2	1995
2	1	1963
. . . . .		

logement			
1	15	rue haute	Briançon
2	3	rue de Lille	Lyon
. . . . .			

"habite" contient l'information suivante

"la personne n°1 habite le logement n°2 depuis 1995, la personne n°2 habite le logement n°1 depuis 1963".

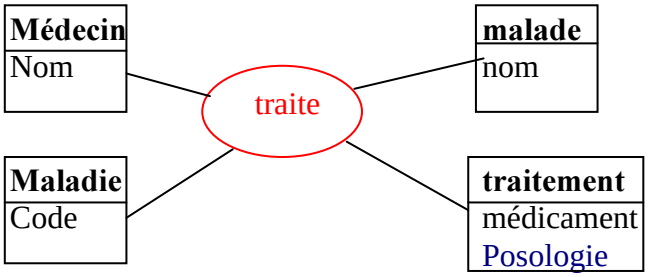
Donc la collection de l'association "habite" reliant Personne(nom, prénom) et domicile(n°, rue,ville) est:

Nom	prénom	n°	rue	ville	date
Martin	jean	3	de lille	Lyon	1995
Dupont	sophie	15	haute	Briançon	19

2.1.2.5. Associations n-aires

On rencontre le plus souvent des associations binaires, mais plus généralement, on rencontrera des associations "n-aire" entre plus de deux entités. On appelle "arité de l'association son nombre de "pattes"

Exemple: soins médicaux:; on a une association dont "l'arité" est 4 entre un "médecin", un "malade", une "maladie", un "traitement":

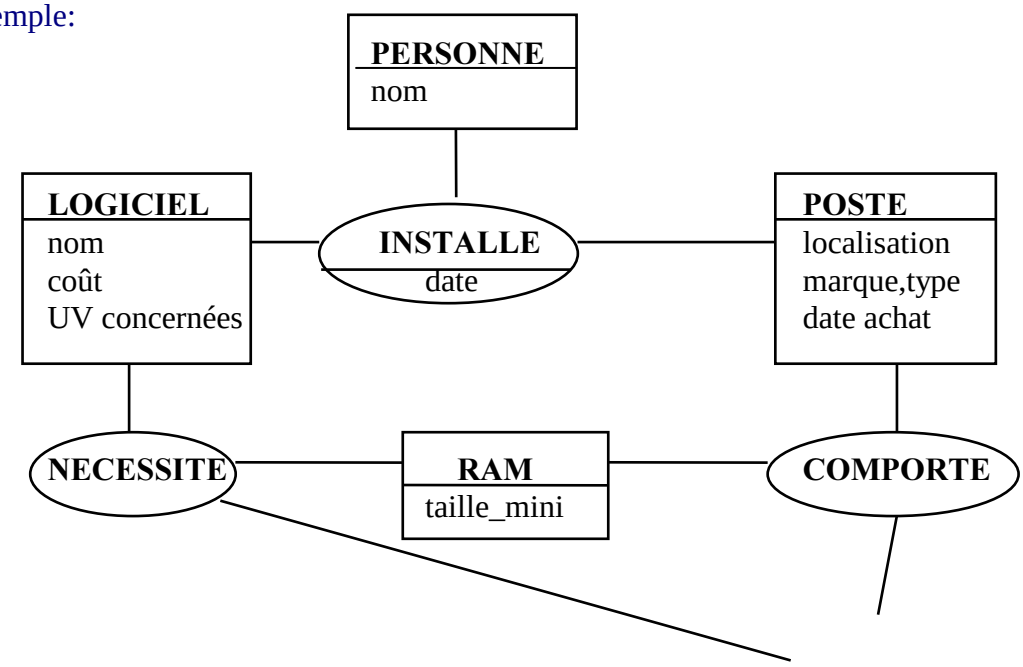


On dira ainsi "qu'un médecin traite un malade avec un médicament pour une maladie"

Nota: *Ceci est un cas assez rare, en général on pourra décomposer une association complexe en plusieurs associations binaires*

2.1.2.6. Généralisation

Un MCD peut comporter un nombre quelconque d'entités et d'associations les reliant. Exemple:



DISQUE
capacité

2.1.4. Requêtes

Connaissant les entités et les associations, on pourra programmer des requêtes (ou demandes d'informations); par exemple demander quels sont les parents d'un enfant donné, quels sont les parents ayant plus de 3 enfants, quels sont les produits les plus achetés, etc…

Nota: Les requêtes peuvent porter sur un nombre quelconque d'entités ou associations, par exemple ci dessus une requête recherchant quel est l'imbécile qui a installé un logiciel nécessitant 128 K de RAM alors que l'ordinateur n'en comporte que 32 K

2.1.5. Liens entité-association.

Par les liens, on peut décrire la complexité de l'association.

Exemple 1 une personne a exactement une seule mère, une voiture a exactement un seul constructeur, etc…

Exemple 2 un homme a zéro ou une femme,

Exemple 3 une personne a zéro ou N enfants.

Exemple 4 un client achète 1 ou N produit (s'il en achète 0, ce" n'est pas un client)

L'exemple 3 montre bien que ces considérations vont influer sur la complexité: le problème n'est pas le même en France ou la polygamie est interdite, et ailleurs où elle est autorisée.

2.1.5. Cardinalité du lien:

A chaque lien entre une entité E et une association A on associe sa "cardinalité" formé de deux informations:

La première indique le **nombre minimum de fois que E intervient dans A**. On ne distingue que deux cas, le minimum est 0 ou 1.

Dans l'exemple 1 le minimum est 1

Dans l'exemple 2 le minimum est 0

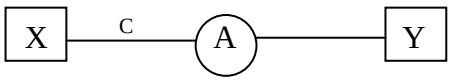
La deuxième indique le maximum: on ne distingue que deux cas

Le maximum est 1 (exemple 1: un homme a au maximum une femme)

Le maximum est supérieur ou égal à 1 mais indéfini (Exemple (2) un homme a au maximum N enfants)

Il y a donc 4 cardinalités possibles pour un lien, notées (0,1) (1,1) (0,n) (1,n) à côté du lien concerné.

Interprétons ces cardinalités et leur impact sur la complexité

Soit la structure 

et étudions l'influence de la cardinalité C du lien X-A

2.1.5.1. les cardinalités 01 et 11

2.1.5.1.1. cardinalité (0,1) signifie que X est relié une fois ou pas du tout à Y par A Une requête du genre "à quoi est relié X?" donnera éventuellement une réponse "vide".

2.1.5.1.2 Cardinalité (1,1): signifie que connaissant une occurrence Xi on connaît automatiquement son correspondant Yi (exemple, connaissant un enfant, on peut déterminer à coup sur, et de manière unique sa mère). On dit que Y est fonction de X, ou qu'il y a une "dépendance fonctionnelle" (DF) entre X et Y

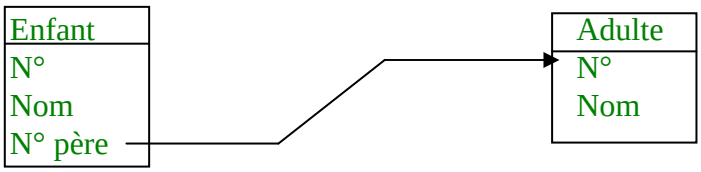
> Remarque 1: si la cardinalité du lien Y-A est également (1,1), on peut dire que X détermine Y et Y détermine X, il y a bijection entre X et Y. En fait X et Y sont une seule et même chose, les deux entités peuvent être fusionnées.

2.1.5.1.3. NOTA: Propriétés communes à (0,1) et (1,1) simplification du MLD

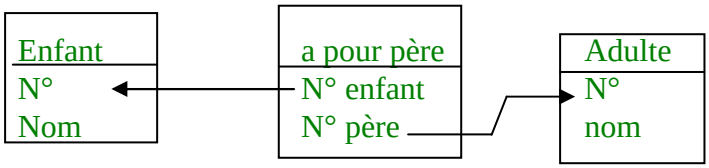
Dans les deux cas, ce type de cardinalité permettra de simplifier considérablement le MLD: Sachant que la connaissance de Xi permet de déterminer au plus un Yi, l'adresse de Yi peut être directement introduite dans l'entité X

Exemple:

deux cardinalités: (1,1) entre l'enfant et "a pour père", (0,n) entre un adulte et "a pour père"
la réalisation pratique devra faire abstraction de la table de l'association en mettant directement le N° de père dans la table "enfant:



Au lieu de:

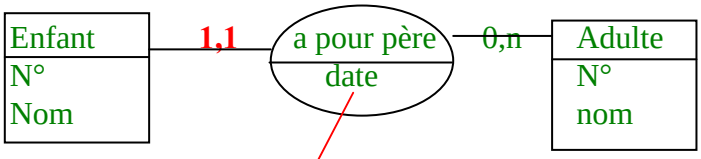


Le fait d'avoir une cardinalité (1,1) pourra donc éventuellement simplifier la réalisation

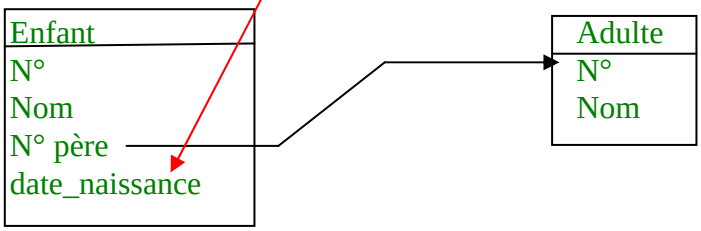
Remarque:

dans ce cas, si l'association comporte des **champs propres,** ils doivent aussi être déplacés.

Exemple:



donnera le MLD suivant:



2.1.5.1.4 Application: clef ou identifiant relatif

On peut dans les cas de cardinalités (0,1) ou (1,1) utiliser une clef relative: un enfant peut par exemple être repéré par

a) la clef externe "n° de père"
b) une clef relative "n° d'ordre dans la famille"

2.1.5.2. Cardinalités (0,n) et (1,n):

Dans les deux cas cela signifie que pour un Xi donné, il peut y avoir plusieurs Yij correspondant. Deux conséquences:

a) Il n'est donc pas question de mettre les identifiants de ces Yij dans X , la table de A est indispensable.
b) La recherche des Yij correspondant à un Xi donné supposera l'examen de toute la table A, donc un temps de calcul important.

D'une manière générale, on constate donc que l'examen attentif des cardinalités est fondamental.

2.1.6. Une notion importante: les définitions fonctionnelles (DF)

Définition:
On dit qu'il existe une "**dépendance fonctionnelle"** entre un ensemble d'attributs a,b,c…n et un attribut Y si la connaissance de a,b…n entraîne une seule valeur de y ou encore si

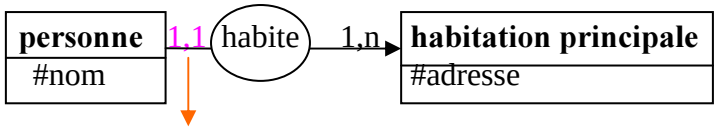
y=f(a,b,c,..n)

On distingue divers types de dépendances fonctionnelle:

a) la dépendance des attributs d'une entité vis à vis de l'identifiant de l'entité.
ces dépendances doivent respecter des règles élémentaires décrites ci dessous.

a) les DF inter entités définies par des cardinalités (1,1).

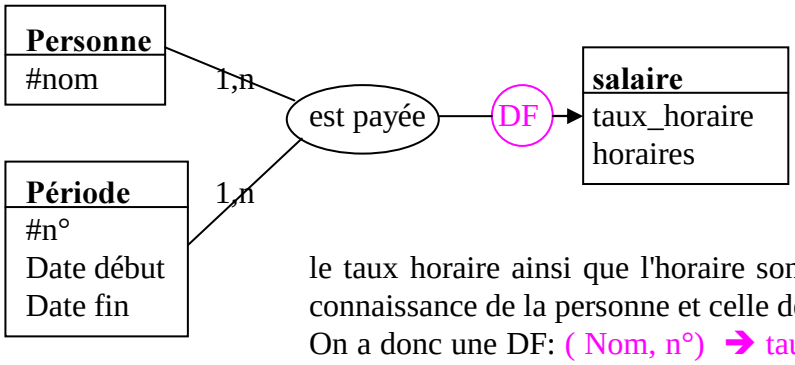
exemple



on a une dépendance fonctionnelle nom➔adresse

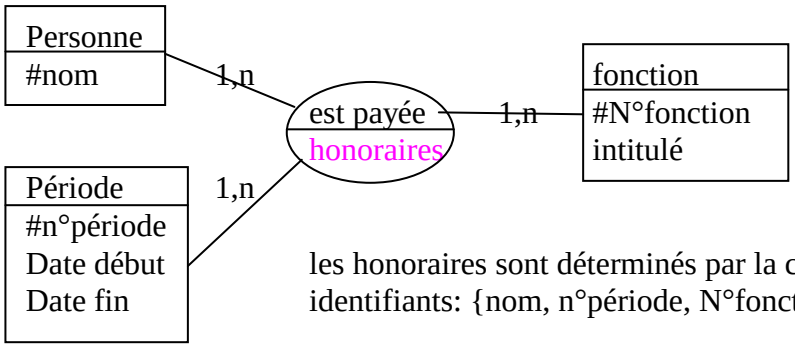
b) les DF désignées explicitement et indépendamment des cardinalités

exemple la connaissance d'une personne et d'une période entraîne la connaissance de son salaire.



le taux horaire ainsi que l'horaire sont déterminés par la connaissance de la personne et celle de la période
On a donc une DF: (Nom, n°) ➔ taux horaire, bien que les cardinalités ne soient pas (1,1)

d) les DF entre des entités et un attribut d'une association
exemple:



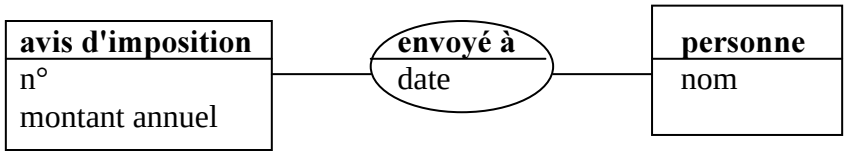
les honoraires sont déterminés par la connaissances des 3 identifiants: {nom, n°période, N°fonction}➔ honoraires.

Notons que si les DF de type (d) sont systématiques, celles de type (c) ne le sont pas et doivent être indiquées

REMARQUE 1:

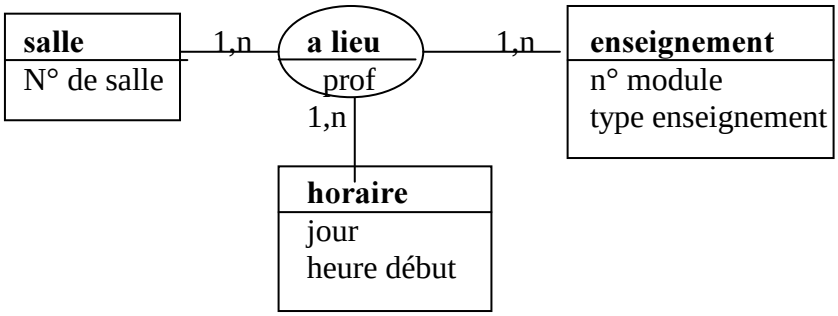
Certaines dépendances semblent ne pas être des DF car la valeur du résultat est inconnu.

exemple 1:



une personne n'a pas forcément fait l'objet d'un avis d'imposition; mais dans la base de donnée des impôts, une telle personne ne figure pas: ne figurent que celles qui ont fait l'objet d'un avis d'imposition. donc quand on connaît la personne et la date, on connaît l'avis; il y a une DF {nom,date} ==>avis.

exemple 2:



Quand on connaît le triplet {N°_de_salle, date_heure, N°de_module}, on connaît le professeur qui enseigne dans cette salle, à ce module, à cette heure. Evidement, si on se donne des valeurs quelconques de ces champs, par exemple {salle=225, date_heure=dimanche à 20 heures, module=programmation} on n'a évidemment pas de réponse. Mais cette valeur du triplet ne figure pas dans la base. Ne figurent dans la base que les triplets **pertinents.** Il y a bien une DF**.**

REMARQUE 2:Ces dépendances fonctionnelles vont jouer un grand rôle dans la normalisation des bases de données:

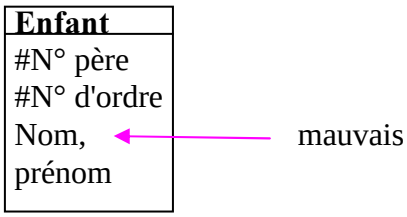
On exigera par exemple que les DF soient "élémentaires" (deuxième forme normale), "directes" (troisième forme normale La présence de DF de type (d) pourra imposer les décompositions des associations (4° et 5° formes normales)

Quelques règles dans la définition des champs d'une entité

Règle 1: toute entité doit avoir une clef, dont dépend fonctionnellement toutes les autres propriétés. Cette clef n'est pas forcément constituée d'un seul champ; par exemple le couple (nom, prénom) peut être la clef dans une entité personne.

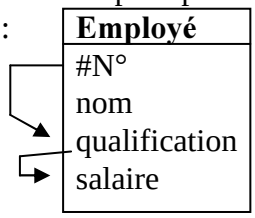
Nota: cela implique la règle suivante, il ne doit jamais y avoir de pluriel dans les champs. Exemple: l'entité **PERSONNE(#N°, nom, prénom, prénoms_des_enfants)** est incorrecte: connaissant le N° de personne on a plusieurs nom d'enfant, il n'y a donc pas de dépendance fonctionnelle. Il faudra donc créer l'association "est enfant de'" reliant un personne et un enfant

Règle 2: tous les champs dépendent fonctionnellement de la totalité de la clef.
Exemple: une entité enfant d'un employé d'une entreprise {N° du père, n° d'ordre}



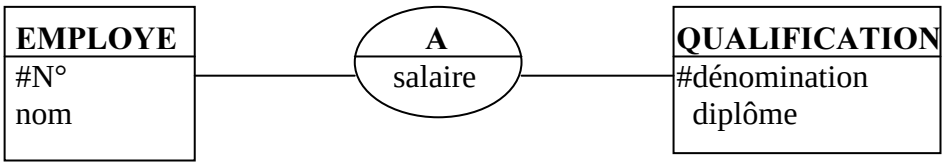
Pour identifier cet enfant, on a utilisé le couple {N° d'employé, n° d'ordre de l'enfant}. c'est mauvais: le nom de l'enfant ne dépend pas de son N° d'ordre, mais uniquement du n° du père. En fait le nom n'a rien à faire ici, on pourra le déduire du n° d'employé. Par contre, le prénom dépend bien du couple (N° père, N° d'ordre): l'enfant N°3 de l'employé N° 15 s'appelle Pierre.

Règle 3: les champs dépendent directement de la clef.
Exemple:



C'est mauvais parce que le salaire dépend non seulement du nom mais aussi de la qualification qui elle peut changer

Il faudra créer une entité supplémentaire qualification(dénomination, diplôme,…) et une association entre qualification et employé ; par exemple:

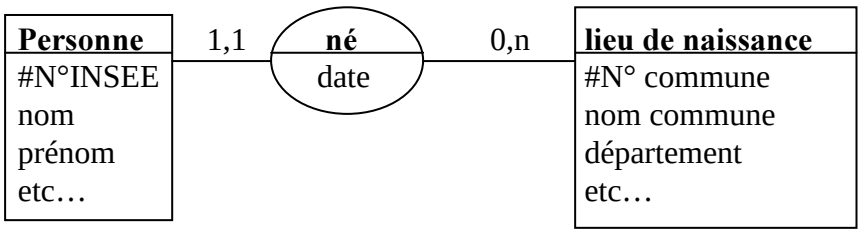


Cette fois c'est correct, il y a une DF (N°, dénomination) ➔ salaire.

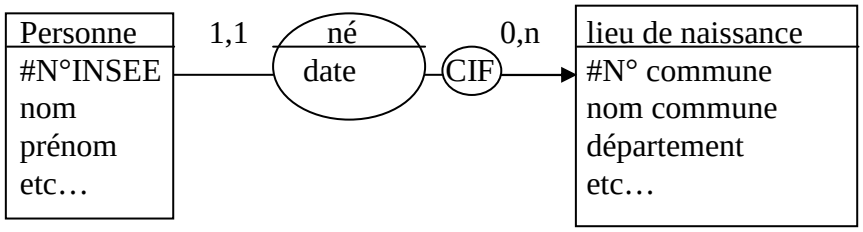
2.1.7. Contraintes d'intégrité fonctionnelle (CIF)

Une CIF est une dépendance fonctionnelle **permanente** entre deux entités. Cela implique que certains identifiants ne peuvent pas être modifiés ni supprimés.

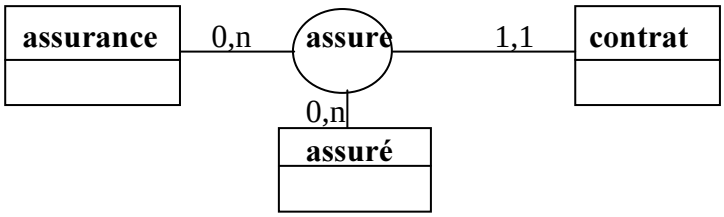
Exemple:



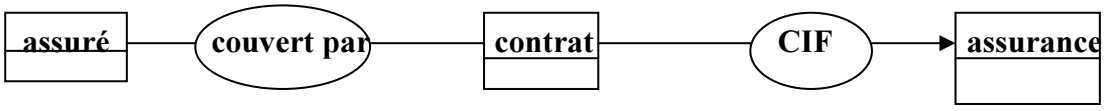
Il y a ici une DF: tous les champs de lieu de naissance sont dépendants de N°INSEE; mais de plus, il y a une CIF {N°INSEE➔lieu de naissance}. cela veut dire qu'on ne peut modifier ni supprimer une propriété de "lieu de naissance " sinon la base ne sera plus cohérente. Par contre, on pourra supprimer une personne de la base, puis son lieu de naissance. Une CIF doit être indiquée spécialement, elle ne se voit pas sur les seules cardinalités



Nota: la connaissance des CIF permet de simplifier certains MCD.
Exemple:



on sait qu'un contrat n'est signé que par une seule compagnie d'assurance (c'est invariable): il y a une CIF (contrat ➔ assurance) On peut alors simplifier l'association "assure" en la décomposant:



remarque: il n'y a pas de CIF entre assuré et contrat.

2.1.8 . ETUDE DE CAS

Système de facturation:

a) solution "classique":un seul fichier (type EXCEL) pour les factures

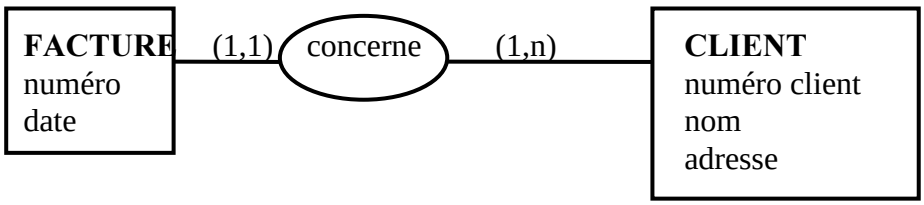
facture= {nom_client, adresse_client, date,
produit1, quantité1, prix_unitaire1, prix_total 1,
produit2, quantité2, prix unitaire2, prix total 2,
etc... ,
total hors taxe, TVA, total TTC}

défauts:

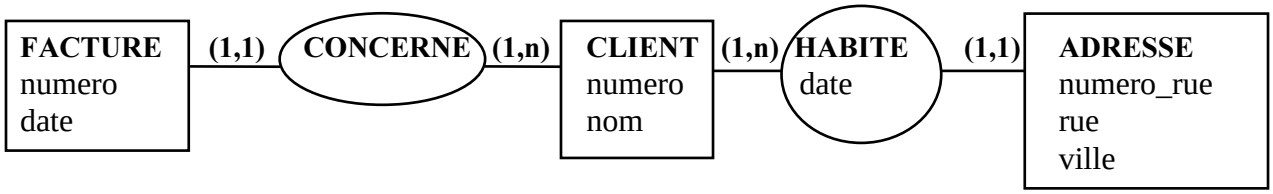
- les noms et adresses clients seront multipliés (pour chaque facture d'un même client, l'adresse est répétée)..
- Il sera difficile de suivre un client qui change souvent d'adresse.
- Le nombre de produits par facture n'est pas borné (si on suppose par exemple 50 produits maximum, les champs seront en général vides, si on met beaucoup moins, on sera obligé de faire plusieurs factures pour des achats groupés.

b) amélioration du cas "client"

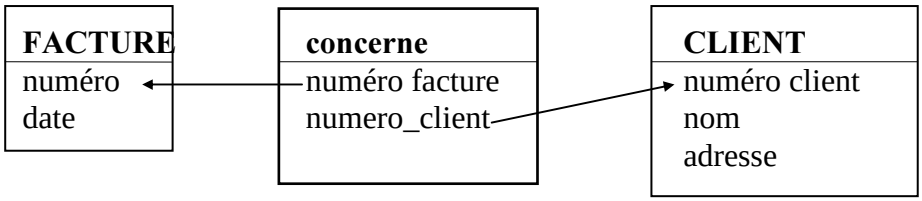
MCD qui suppose que l'adresse est constante:



si l'adresse est variable



correspondance avec le modèle logique de données. En ACCESS par exemple: *Concerne* devient une table avec deux pointeurs , l'un vers la facture, l'autre vers le client.

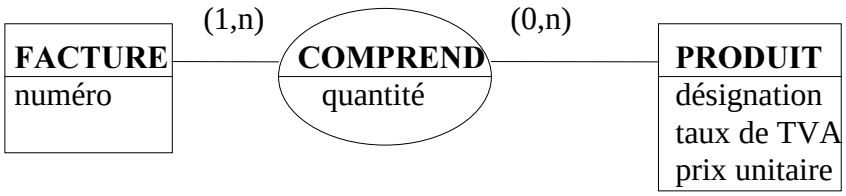


cas des produits

Règle quand on a un "pluriel", on fait intervenir une relation; exemple de définition: facture={nom_client, adresse_client, date, produits }:

Le problème est de savoir répartir les informations entre les entités.

a) hypothèse 1: les prix des produits et le taux de TVA ne varient pas dans le temps



nota: on peut faire deux sous hypothèses

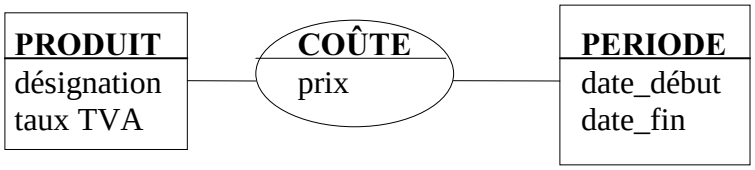
- Ou bien le calcul du total de la facture se fait rarement (en principe une fois seulement); on prévoit de disposer d'une "requête" d'affichage qui calcule successivement:

  prix HT du produit= prix_unitaire * quantité
  total HT= ∑ prix HT
  TVA= total HT * taux_TVA
  prix TTC= Total HT + TVA

Les champs total_HT, TVA, prix_TTC, ne figurent pas dans la facture ➔ gain de place

- Ou bien on doit recalculer très souvent le montant, il vaut mieux prévoir des champs: total_HT, TVA, prix_TTC, etc... ➔ gain de temps.

b) **hypothèse de variabilité des prix dans le temps.**
Il y a plusieurs prix pour un même produit, on introduit une relation "coûte"



établir le montant d'une facture comportera outre les opérations ci dessus, la recherche de la période encadrant la date de la facture.

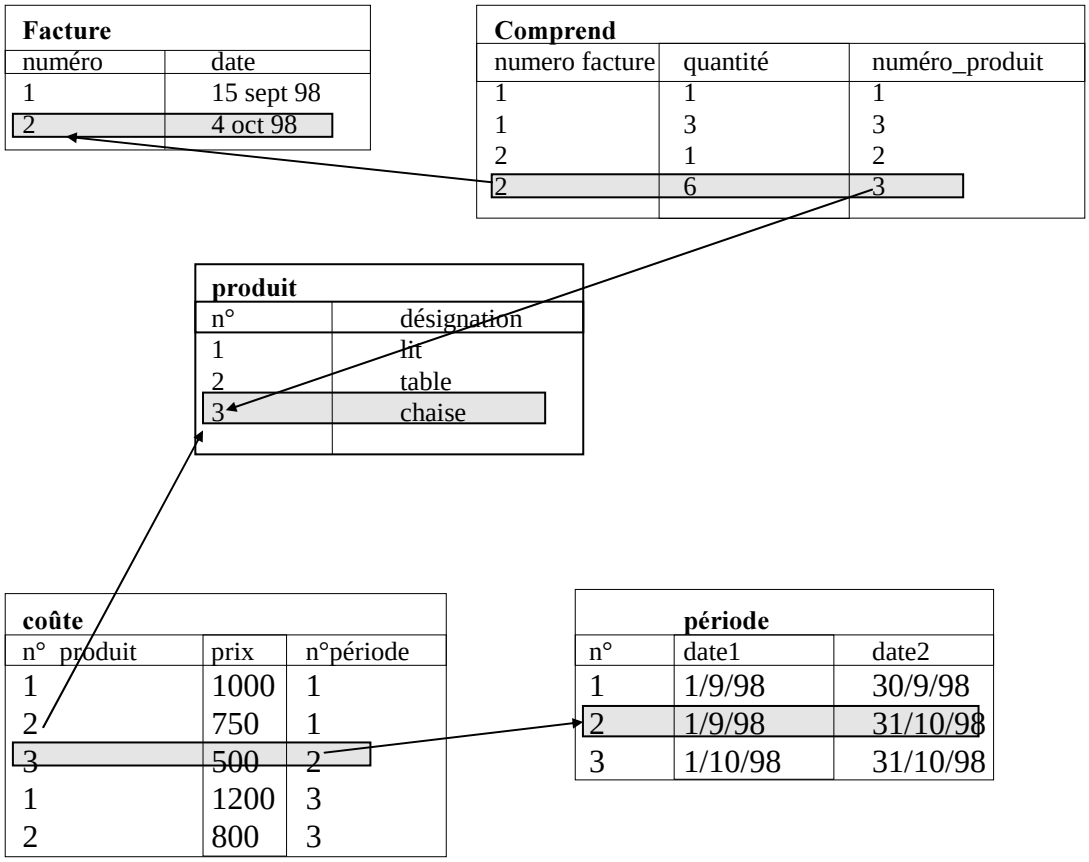
Exemple pratique: deux factures

facture1: 15 septembre 1 lit et 3 chaises
facture 2 4 octobre 1 table et 6 chaises

prix:

en septembre: lit 1000F, table=750F, chaise= 500F
en octobre: lit 1200F, table=800F, chaise= 500F

contenu des tables:



Facture

numéro	date
1	15 sept 98
2	4 oct 98

Comprend

numero facture	quantité	numéro_produit
1	1	1
1	3	3
2	1	2
2	6	3

produit

n°	désignation
1	lit
2	table
3	chaise

coûte

n° produit	prix	n°période
1	1000	1
2	750	1
3	500	2
1	1200	3
2	800	3

période

n°	date1	date2
1	1/9/98	30/9/98
2	1/9/98	31/10/98
3	1/10/98	31/10/98

les flèches montrent par exemple que
la facture n°2 du 4 octobre comporte 6 produits n° 3 (chaises) dont le prix en octobre est de 500F

Estimation des coûts. Exemple grande surface

Par an:
Nombre de factures N=1 000 000
Nombre maximum de produits par facture (dans solution classique): P=100
Nombre de produits différents:10 000
nombre moyen de produit par facture: 20
nombre de périodes de prix: 10

coût de la solution 1 (uniquement dans la partie produits):

une ligne de facture "papier" ferait environ 100 caractères couvrant désignation, quantité, prix unitaire, prix hors taxe, etc...

1000000* (coût d'une facture)= 1 000 000 * (100 * 100) = 10 giga octets

coût de la solution 2

table "facture": n°=entier long: 4 caractères
date:3 entiers 6 caractères
1 000 000 factures
total 10 M octets

table "comprend"
numéros de facture 4caractères
n° produit entier 2 caractères
quantité=entier 2 caractères
nombre d'achats 20* 1 000 000
total 16 0 Moctets

table "produit"
n° produit: entier 2 caractères
désignation 100 caractères
nombre de lignes: 1000
total 200 000 octets: négligeable

table "coûte"
n° produit 2 caractères
prix (réel) 4 caractères
n° période 2 caractères
nombre de lignes 10 000 *10 (: chaque produit peut changer 10 fois de prix)
total 800 000 octets

table "période" (on suppose que chaque prix de produit a sa propre période de validité !)
n° période entier 2 octets
2*(date= 3 entiers) 12 octets
nombre de périodes 133000 (=365*365)
total 1.8 Mega octets (très surestimé !)

TOTAL solution 2 200 Mega octets

REMARQUE division par 50 du coût total !

Quelques extensions à MERISE

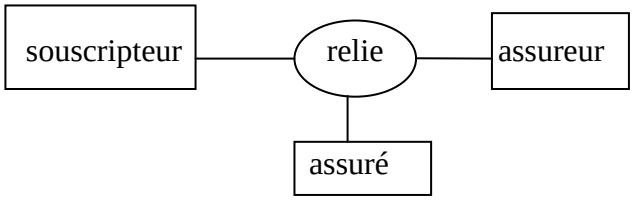
1. Associations d'associations

1.1: remarques préliminaires:

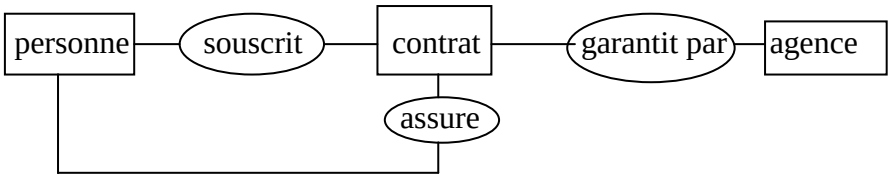
Le modèle Merise distingue clairement la notion d'entité de celle d'association. Pourtant la distinction n'est pas aussi évidente qu'il y paraît:

a) souvent les rôles des deux peuvent être permutés. Exemple:

un contrat d'assurance lie un assuré, un assureur, un souscripteur:



mais on pourrait dire aussi:
un contrat est souscrit par une personne. Une agence garantit le contrat. Un contrat assure une personne:



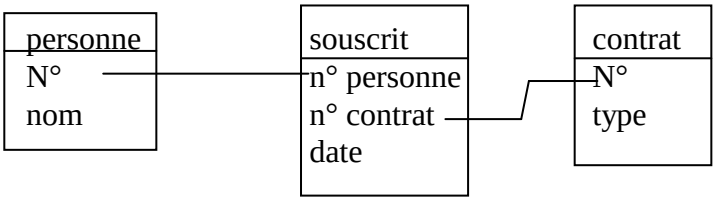
ici le rôle central n'est plus l'association "relie" mais l'entité "contrat", les entités "souscripteur" et "assuré" sont remplacées par des associations "souscrit" et assure".

On constate donc une certaine dualité entre entité et association.

Ceci traduit d'ailleurs le fait que la langue naturelle permet ce genre de dualité: Exemple: "les chats mangent les souris" peut se transposer en "le fait de manger est effectué par ceux qui sont des chats et subi par ceux qui sont des souris" : on intervertit les rôles de groupe nominal et groupe verbal !

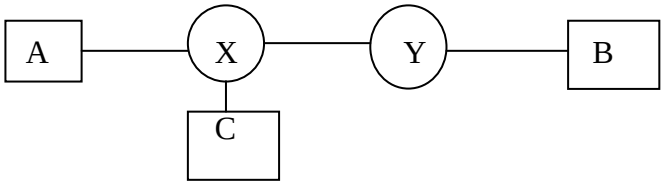
b) quand on passe au MLD, par exemple en ACCESS, la différence entre entité et association disparaît, les deux sont des tables. Exemple:

donne comme MLD:

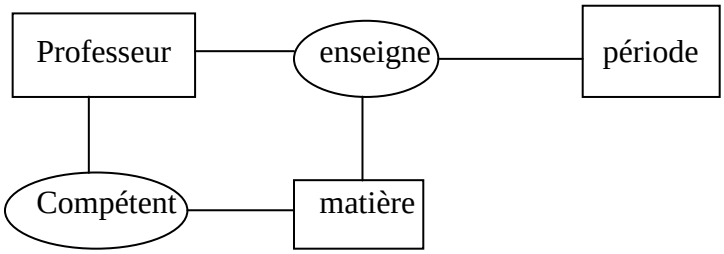


On emarque que les liens entre tables ne sont pas "orientés", ils signifient simplement que les deux champs aux extrémités doivent être égaux (l'égalité est une relation réflexive!)

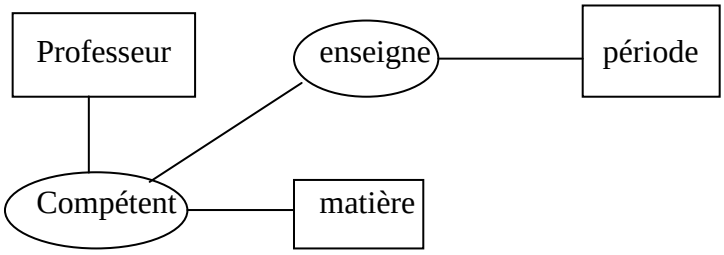
1.2. Corollaire: Les deux remarques ci dessus montrent que la différence entre entité et association est finalement minime; donc si un lien peut exister entre une entité et une association, il peut aussi exister entre deux entités. Rien ne s'oppose donc à ce qu'on fasse des associations d'associations: par exemple,



1.3. Exemple d'application:



Ce MCD est **mauvais:** rien n'empêche d'affecter à un enseignement un prof incompétent! Il est préférable de choisir pour un enseignement **un couple existant** {prof, compétence}

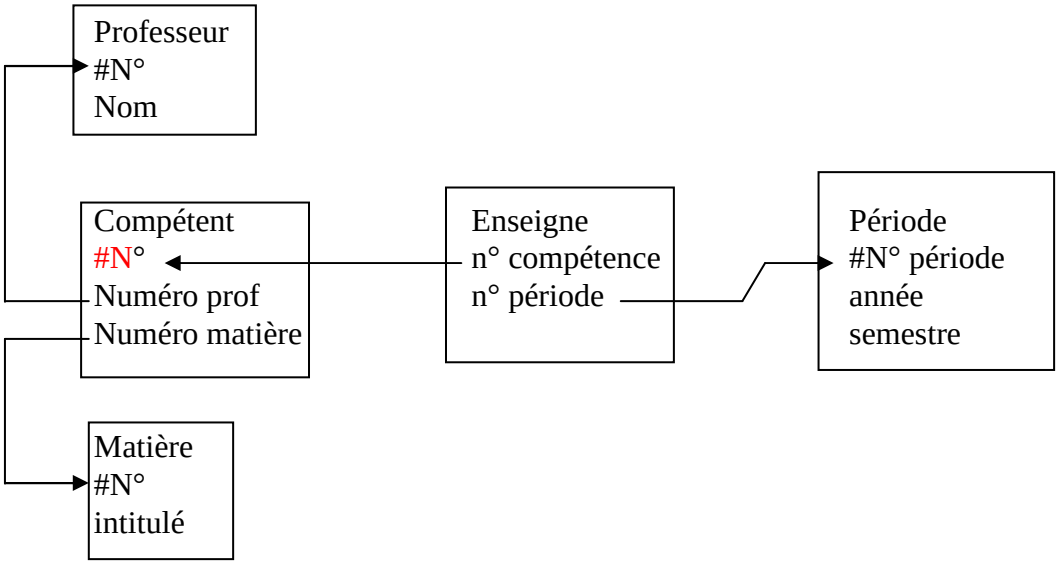


Ici à la saisie des données, il sera impossible d'affecter à un enseignement un couple inexistant. (Evidement cela n'empêche pas de déclarer compétent quelqu'un qui ne l'est pas, mais c'est une autre histoire!)

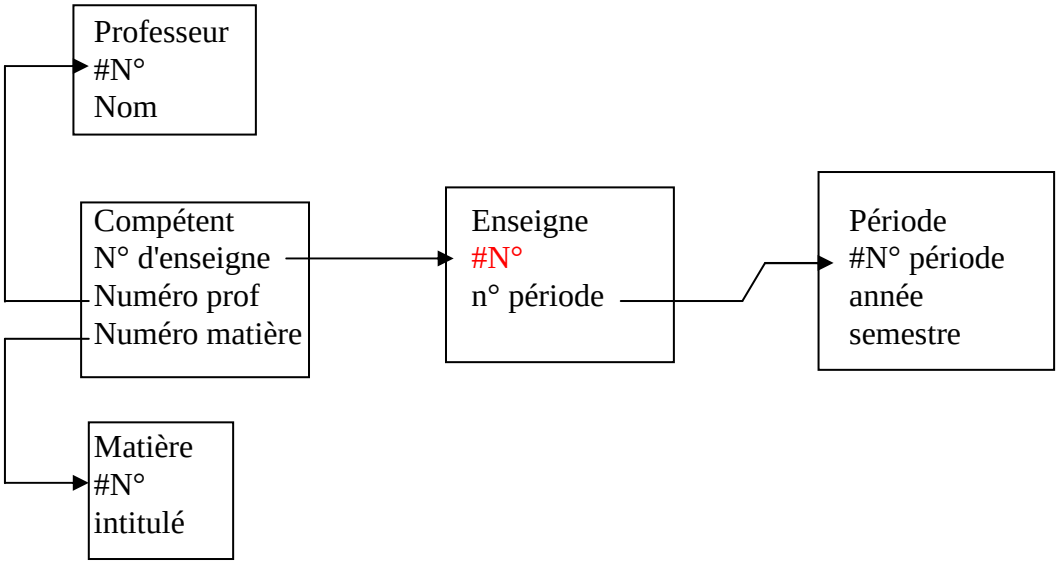
1.4. orientation du lien entre associations.

Ici le lien {compétent – enseigne} est orienté: il y a deux MLD possibles:

Modèle 1: toutes les compétences sont numérotées, dans la table enseigne on trouve un "pointeur" sur la table compétence.

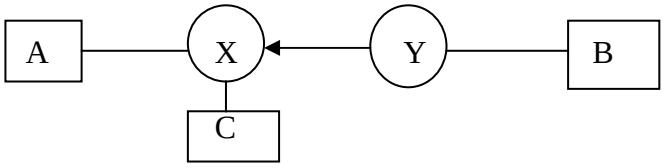


Modèle 2 tous les enseignements sont numérotés; dans la table compétent on trouve un pointeur sur la table enseigne:



Les deux modèles ne sont pas tout à fait équivalents: en particulier on peut dire que l'enseignement implique la compétence, mais pas l'inverse (il y a des gens compétents qui n'enseignent pas!) donc à priori, la clef primaire est dans "compétence", la clef secondaire dans "enseigne".

Notation: le sens du pointeur est indiqué par une flèche: exemple



2) héritage (notion importée de la "programmation orientée objet").

On rencontre très fréquemment des entités qui se ressemblent; exemple:

Professeur(nom, prénom, date_naissance, adresse, discipline) et
Etudiant(nom, prénom, date_naissance, adresse, études)

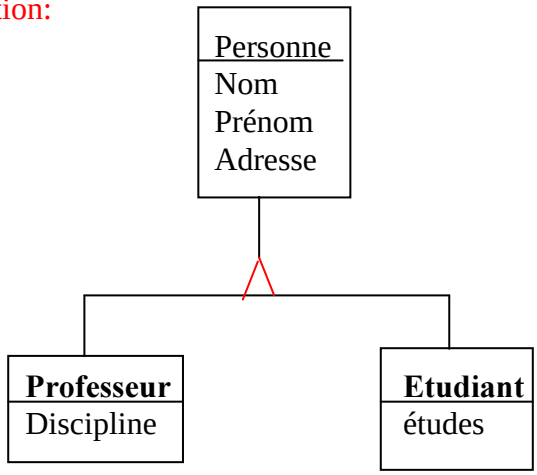
Une telle situation est absurde: beaucoup de fonctionnalité concernant le professeur et l'étudiant vont être dédoublées (par exemple la fonction "envoi de courrier")

Solution:
a) on va faire une sorte de "'mise en facteur" de la partie commune, en créant une entité nouvelle:

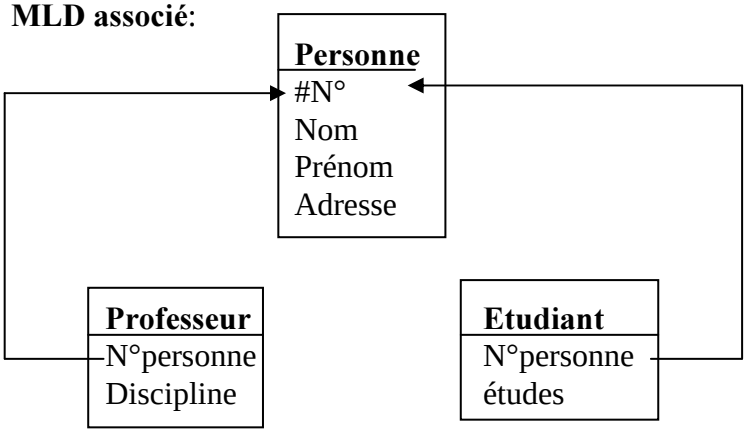
Personne(nom, prénom, date_naissance, adresse)

c) on dit que un professeur est une personne avec en plus une discipline
d) on dit qu'un étudiant est une personne avec en plus des études.

Notation:



MLD associé:



Ainsi toutes les fonctionnalités concernant les parties communes ne seront écrites qu'une fois (par exemple la fonction de publi-postage).

NOTA: on pourrait imaginer des pointeurs dans l'autre sens, mais
a) cela interdirait à un professeur d'être en même temps étudiant .
b) une personne peut être ni prof ni étudiant.

Remarque

La notion d'héritage est beaucoup plus riche que ce qui est présenté ici:

a) On peut envisager des "héritages multiples" (par exemple l'entité "moto" héritera simultanément de l'entité "véhicule à moteur" et de l'entité "deux roues").

c) on peut considérer des héritages pour les associations, pour les liens

d) on peut voir apparaître des contraintes d'héritage; par exemple on pourra imposer qu'une personne soit un prof ou un étudiant, mais pas les deux, éventuellement aucun, ou qu'une personne soit forcément soit homme soit femme, ...

Se reporter aux ouvrages sur la POO (programmation orientée objet).

CHAPITRE 3
Conception de Bases de données

3.1. Introduction

Le problème est complexe, et il n'y a pas de méthode générale assurant que la conception est correcte. On peut néanmoins dégager des grandes catégories de méthodes..

Méthodes intuitives:
Méthodes rigoureuses

Les méthodes intuitives visent à définir une architecture de la base à partir des connaissances générales obtenues par dialogue avec l'usager futur. Deux directions sont possibles

3.1.1. on peut partir des "requêtes" souhaitées par l'usager.

Exemple: Base de donnée du parc de matériel informatique:

L'utilisateur nous dit qu'il veut disposer des requêtes suivantes:

- connaître la composition de son ordinateur (capacité RAM et Disque, type UC, type d'écran, etc...)
- savoir l'état d'une machine (âge, pannes, dates de maintenance,...)
- savoir où elle est située
- savoir quels sont les logiciels installés

etc..

On voit donc apparaître les champs utiles.

Dans un premier temps on prévoit une entité unique telle que:

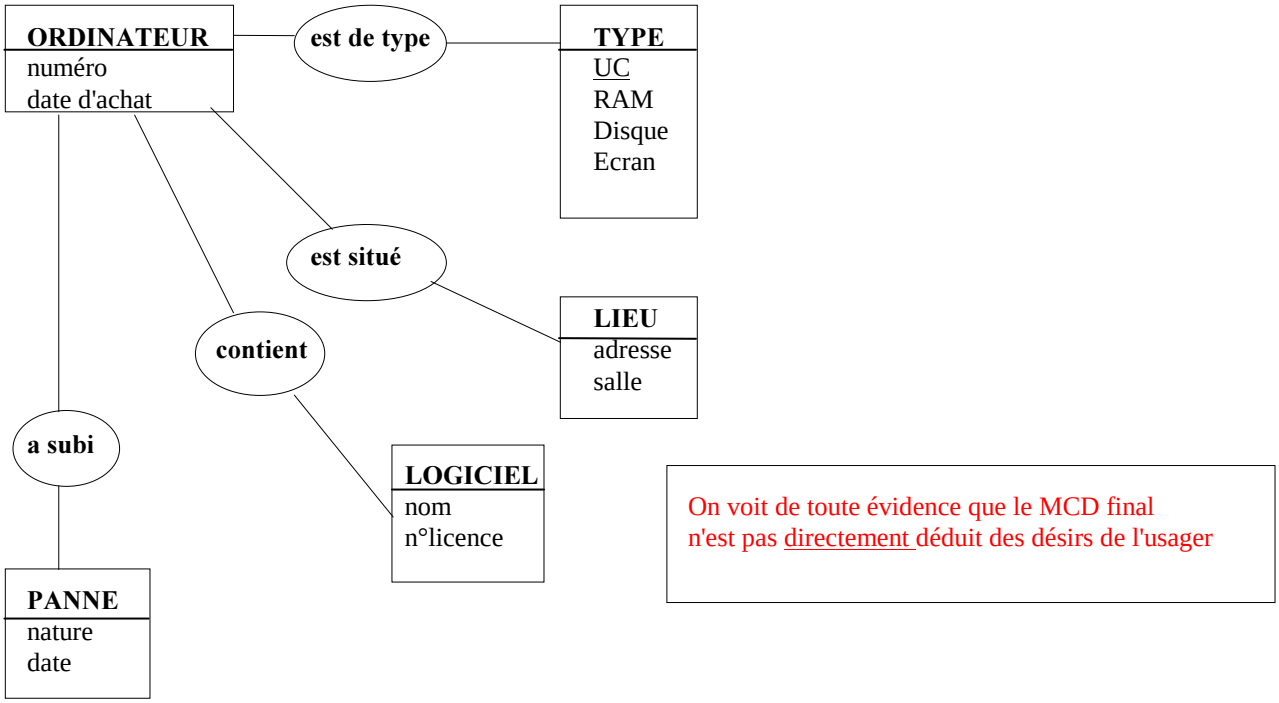
ORDINATEUR
numero
UC
taille RAM
taille Disque
écran
date achat
pannes
maintenances
lieu
logiciels

Cette entité présente de nombreux défauts:

a) il y a des champs avec la marque du pluriel (pannes, logiciels)
On sait corriger cette erreur en introduisant des associations, par exemple "contient" entre "ordinateur" et "logiciel"

b) un autre défaut est de créer autant d'entités qu'il y a de machines, même si certaines sont identiques

on arrive donc par améliorations successives à la structure suivante:



3.1.2. Méthode "sémantique"

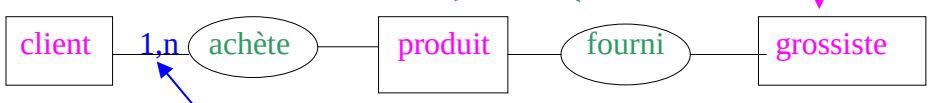
Il est préférable de partir d'une description complète du système dont on décrit les données. Pour ceci, on demande au client de faire une description **en français** de son "système", et on analyse phrase par phrase:

exemple

"un client achète des produits. Ceux ci sont fournis par un grossiste"

on distingue tout de suite les entités **"client", "produit", "grossiste"** (les groupes nominaux de la phrase)
et les associations **"achète"** et **"fournis"** (les verbes de la phrase)

on obtient



Nota:

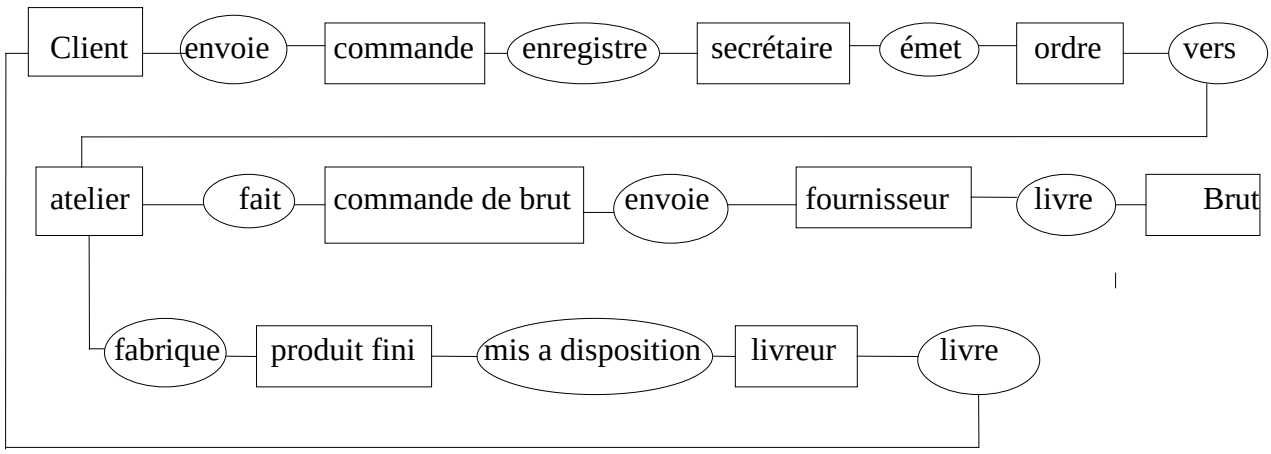
a) on n'a pas encore le détail des champs, mais on a trouvé le plus important: la structure générale du MCD.

b) le texte fourni même les cardinalités des liens, par exemple *"achète des produits"*, donc cardinalité (1,n) entre "client" et "achète".

Limites de la méthode:

Il est bien évident que la description en français sera souvent incorrecte, incomplète, sujette à révisions, etc..

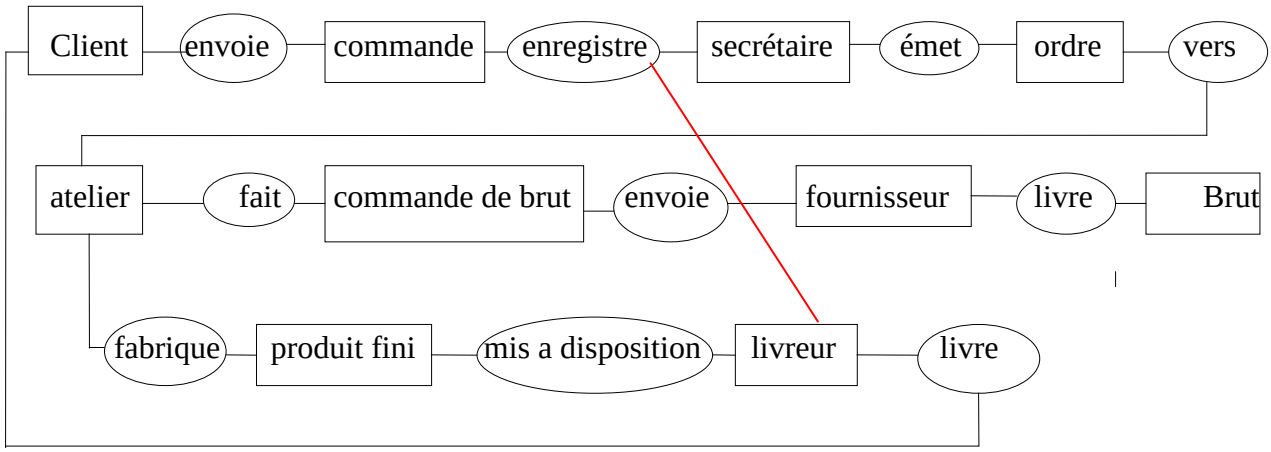
donnons un exemple plus complet (les associations associées aux verbes sont soulignée):
"Quand un client envoie une commande, celle ci est enregistrée par le service commercial qui émet à son tour un ordre de fabrication vers un atelier. Celui-ci fait une commande de produit brut et l'envoie à un fournisseur. Le fournisseur livre le produit brut à l'atelier qui fabriquera le produit fini. Celui-ci est mis a disposition du service livraison qui livrera le client".



Comme ce n'est pas très clair, le MCD a des défauts peu visibles. Par exemple, on constate que le livreur ne saura pas très probablement pas à qui livrer le produit:

Si effectivement, on peut toujours "remonter la chaine", à chaque fois on multiplie les informations: connaissant son nom, le livreur retrouvera tout ce qu'on lui a mis à disposition; pour chacun de ces produits on trouvera l'atelier qui l'a fabriqué; pour chacun de ces ateliers, on retrouvera tous les ordres émis, etc... On constate que chaque fois qu'on passe par un lien de cardinalité (0,n) ou (1,n) on **multiplie la quantité d'informations** extraites de la base
Sauf si le nom du client figure dans chaque entité, il y aura indécision finale.

On a simplement oublié d'envoyer un double de la commande au service livraison! Il faut rajouter un lien entre l'association "enregistre" et l'entité "livreur". (lien en rouge)



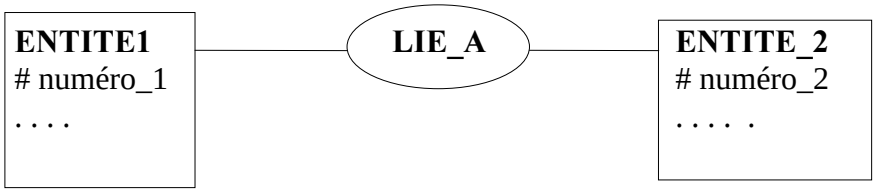
Corollaire

On n'a pas intérêt a avoir des chaînes trop longues sous peine de voir les requêtes de sélection donner beaucoup d'informations non pertinentes.

Il pourra souvent être utile d'introduire des "raccourcis" comme le lien commande=>livreur de l'exemple précédent.

b) intéressons nous au coût en temps de la recherche d'information. Ce coût est fonction cette fois de la taille de tables représentant les associations

exemple



L'association "lié_à" est représentée par une table de n couples {numéro_1, numéro_2}
cette table peut être trié suivants les numéros 1 ou 2, mais pas les deux. Donc le coût de recherche ne sera faible que pour un seul des sens 1=>2 ou 2=>1

3.4. Les erreurs fréquentes:

Deux tendances contradictoires: faire trop compliqué et faire trop simple.

3.4.1. Faire trop compliqué:

L'erreur la plus fréquente est de définir trop d'entités correspondants à des concepts voisins.

Exemples:

a) définir deux entités "client" et "fournisseur", alors que les champs sont probablement de nature très peu différente, voire identique. On définira plutôt une entité "entreprise" qui groupera les deux.

b) Définir des entités "commande", "devis" et "facture" très semblables, au lieu de les grouper en une seule entité "document" . Ces entités comporteront un champ supplémentaire "type de document".

Remarque Parfois des entités **se ressemblent mais ne sont pas identiques.** On utilisera alors la notion d'héritage (voir plus loin) .

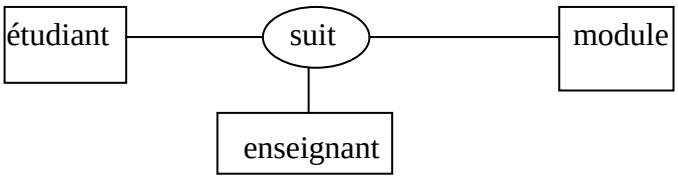
3.4.2. faire trop simple (simple au niveau du modèle peut conduire à un coût final important)

Erreur très fréquente: grouper dans une même association des relations qui sont indépendantes.

Exemple de texte:

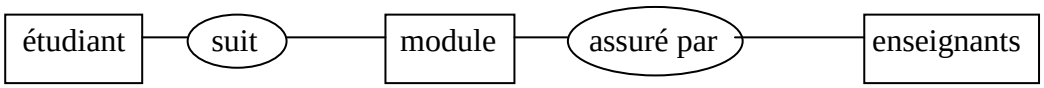
"un étudiant suit des modules avec des enseignants"

ce texte conduit à la structure suivante:



Mais cette structure est mauvaise, la relation enseignant/étudiant n'est pas pertinente. La phrase correcte aurait dû être:

"un étudiant suit des modules qui sont assurés par des enseignants"



On dit qu'on a **projeté** l'association suit(étudiant, enseignant, module) en deux associations suit(étudiant, module) et assuré_par(module, enseignant).

Voir également au paragraphe "projection d'une association".

L'intérêt est double:

a) au niveau de l'encombrement mémoire:

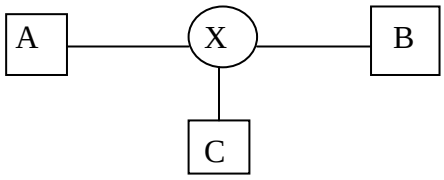
- Dans la première solution, il y a un pointeur vers l'enseignant pour chaque étudiant.
- Dans la deuxième solution, dans "assuré par" on trouve certes deux pointeurs, mais il y a une occurrence de ce couple pour chaque module et non pour chaque étudiant . Le gain de place est important.

b) au niveau des temps de mise à jour :

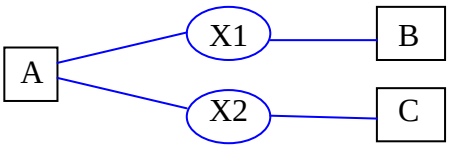
- dans la première solution, remplacer un prof par un autre, implique une modification pour chaque étudiant,
- dans la deuxième, il y a une seule modification à faire.

Généralisation: Projection d'une association

Considérons une association ternaire de la forme
Notée X(A,B,C)

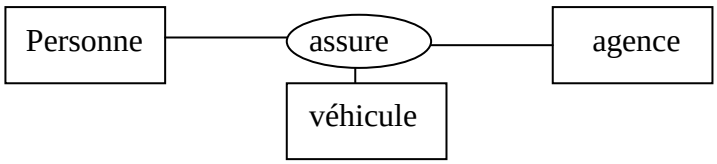


Cette association peut être "projetée" suivant deux seulement des entités: Projection de X(A,B,C) sur A et B, notée X1(A,B) et projection de X(A,B,C) sur A,C notée X2(A,C).



Le problème est de sa voir si ces projections redonnent les mêmes informations que l'association initiale. **En général la réponse est NON**, toute association ternaire n'est pas forcément équivalente à deux associations binaires.

Exemple 1



supposons que la requête "qui assure quoi?" donne les résultats suivants:

personne	agence	véhicule
Jean	Axa	256 ZQ 38
Pierre	Macif	256 ZQ 38
Jean	Macif	312 XS 85

(Il y a deux occurrences de 256 ZQ 38 car Jean a vendu sa voiture à Pierre)
Projetons:

Personne	**Agence**
Jean	Axa
Pierre	Macif
Jean	Macif

Personne	**véhicule**
Jean	256 ZQ38
Jean	312 XS 85
Pierre	256 ZQ38

De ces informations, on pourrait en déduire aussi bien:

"jean a assuré le véhicule 256 ZQ38 à la Macif"

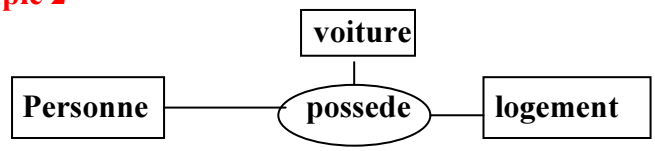
que

"jean a assuré le véhicule 256 ZQ38 chez AXA"

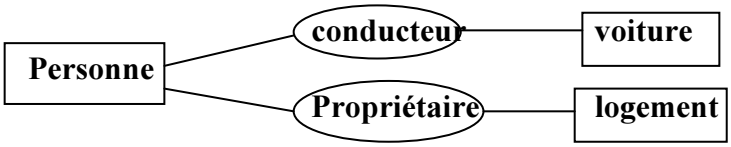
On a manifestement perdu de l'information en projetant

Parfois, la projection est possible: exemple 2

Possède(personne, voiture, logement)



Dans ce cas, il est clair que la voiture n'est pas déterminée par le couple {personne,logement}, mais par la seule personne; de même le logement n'est pas déterminé par le couple {personne,voiture}; la relation peut certainement être décomposée (projetée) sans perte d'information:



REMARQUE 1 Il faut distinguer deux notions:
a) association **"occasionnellement projetable**" (on le constate à l'instant t)
b) association **"toujours projetable"** (on le prouve)

Exemple: L'exemple de l'assurance peut faire croire à une projection possible: considérons la situation "après que Jean ait vendu sa voiture à Pierre", mais "avant en avoir racheté une autre et changé d'assureur":

personne	agence	véhicule
Jean	Axa	256 ZQ 38
Pierre	Macif	256 ZQ 38

Les projections donneront :

Personne	Agence
Jean	Axa
Pierre	Macif

Personne	véhicule
Jean	256 ZQ38
Pierre	256 ZQ38

Il n'y a aucune ambiguïté. On pourrait en déduire "au temps t" que l'association est projetable. Mais dès que Jean rachète une voiture, la base est à refaire!

Par contre dans l'exemple 2 on démontre que c'est projetable (et que ce le sera toujours)

REMARQUE 2 intérêt de la projection:

Dans le cas où c'est possible, le gain en place mémoire peut être considérable en projetant: supposons une base de n gens "riches" ayant en moyenne p voitures et q logements:

Il y a n.p.q occurrences de "possède" dans la version ternaire

n.p occurrences de conducteur et n.q occurrences de "propriétaire"

(exemple n=10000, p=q=10 ➔ n.p.q=1 000 000 alors que n.p+n.q=200 000

CHAPITRE 4
Le langage SQL
(VERSION ACCESS)

SQL est essentiellement un langage d'interrogation de bases de données; son but est de permettre d'extraire de la base des informations pertinentes, accessoirement de les modifier, les supprimer, etc...
SQL est un langage qui manipule des ensembles de données. son usage implique donc une bonne connaissance de la **théorie des ensembles.**

Ce n'est pas un langage de programmation à proprement parler, beaucoup de fonctionnalités manquent. Pour pallier aux manques, en général on accédera aux primitives SQL via un autre langage (Basic par exemple).

En particulier SQL ne peut enchaîner plusieurs instructions; si un traitement exige un tel enchaînement, il devra être fait appel à un langage classique.

1. Instruction SELECT

Cette instruction a pour but de définir un sous ensemble d'un ensemble de données contenues dans une ou plusieurs tables.

1.1. Forme générale:

select ***{liste de champs}*** **from** ***{liste de tables}*** **where** ***<condition> ;***

exemple 1: sélection de champs "nom" et "prénom" dans une table unique "personne" avec la condition que le lieu de naissance soit Paris:

```
SELECT nom, prénom FROM personne WHERE lieu_de_naissance='Paris';
```

Sémantique: L'instruction SELECT a pour but essentiel de définir un ensemble de n-uples issus des tables de la liste et vérifiant les conditions spécifiées.

Règle: si l'instruction SELECT est isolée (si elle n'apparaît pas dans une autre instruction) les n-uples sont écrits.

1.2. Liste de champs.

C'est une liste de noms de champs séparés par des virgules. Ces noms peuvent être qualifiés par le nom de la table dont ils sont issus, ou s'il n'y a pas d'ambiguïté, sous forme simplifiée.

Exemple 1

```
SELECT parent.nom, parent.prénom, enfant.prénom FROM parent, enfant....
```

les parents et les enfants ayant un champ de même intitulé, il faut préciser l'origine du champ sinon il y aurait ambigüité.

Exemple 2:

SELECT nom, prénom FROM parent...

Ici, il n'y a aucune ambiguïté, inutile de qualifier les champs.

Nota: SQL tolère les noms de champs composés de plusieurs mots (séparés par une ponctuation: espace, virgule, tiret, etc...) Dans ce cas, le nom doit être encadré de crochets. Exemple personne.[date de naissance].

Il est possible de sélectionner tous les champs d'une table: SELECT * FROM table;

Expressions: on peut former des expressions à partir des champs sélectionnés, exemple: affichage des prix TTC à partir de prix hors taxe :

SELECT prix* 1.206 FROM produit;

ou en combinant des champs issus d'une même table ou de tables diffférentes:

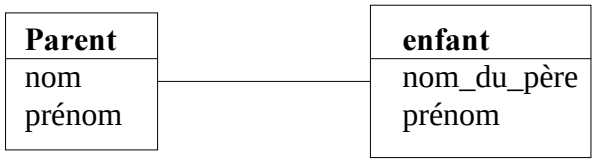
SELECT chambres.prix+ (chambres.prix * TVA.taux) FROM chambres, TVA;

1.3. liste de tables.

1.3.1. Ce peut être une simple suite de noms de tables, liées ou non entre elles comme dans l'exemple 1 ci dessus.

1.3.2. Ce peut être une expression indiquant les liens entre des tables

Exemple3




```
SELECT  enfant.prénom, parent.nom
FROM parent INNER JOIN enfant ON enfant.nom_du_père= parent.nom;
```

Voir également "jointures".

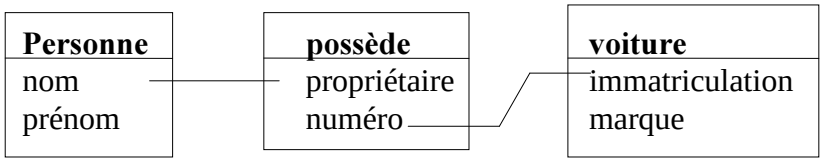
REMARQUE 1: une instruction comme

```
SELECT  enfant.prénom, parent.nom
FROM parent, enfant ;
```

aurait pour résultat d'afficher toutes les combinaisons possibles {enfant.prénom, parent.nom} même s'il n'y a aucun lien de parenté.

Explication: le couple **parent, enfant** désigne le "produit cartésien des deux tables". ce produit cartésien est effectivement calculé, rangé dans une table intermédiaire T (non visible). C'est de cette table que sont extraits les couples enfant.prénom, parent.nom

REMARQUE 2: on rencontrera fréquemment des relations de jointure complexes, en particulier quand des associations relient des entités



SELECT personne.nom, voiture.marque , possède FROM personne INNER JOIN (possède INNER JOIN voiture ON numéro=immatriculation) ON nom=propriétaire;

NOTA: il n'y a pas de limitation à la complexité de la chaîne de relations

1.3.3. La liste de tables peut être une expression définissant dynamiquement un ensemble d'informations.

Exemple 4: Ayant sélectionné le nom et prénom *comme dans l'exemple ci dessus*, on peut en extraire les enfants qui ne s'appellent pas Martin:

Requête_1= SELECT enfant.prénom, parent.nom
FROM parent INNER JOIN enfant ON enfant.nom_du_père= parent.nom

SELECT enfant.prénom, parent.nom FROM [requête_1]
WHERE parent.nom != 'martin';

Le premier SELECT définit un ensemble temporaire (il n'y a pas de création d'une table), dont on extrait les information utiles par le deuxième SELECT. Notons que cet ensemble temporaire peut être mémorisé (voir attribut INTO).

1.3.4. Types de jointure:

Trois types de jointure: INNER, RIGHT, LEFT

exemple Soient les 2 tables suivantes; on veut afficher qui possède quoi.

table personne:

n° personne	nom
1	jean
2	marie
3	sophie
4	pierre

table possède:

n°propriétaire	marque
3	peugeot
2	citroën
	cadillac

a) SELECT nom,marque FROM personne
INNER JOIN posssède ON n°personne= n°propriétaire;

affiche :

marie	citroën
sophie	peugeot

On obtient donc tous les couples {propriétaires - voiture}

b) SELECT nom,marque FROM personne
RIGHT JOIN posssède ON n°personne= n°propriétaire;

affiche :

jean	
marie	citroën
sophie	peugeot
pierre	

On obtient donc tout ce que possèdent les personnes (certaines ne possèdent rien)

c) SELECT nom,marque FROM personne
LEFT JOIN posssède ON n°personne= n°propriétaire;

affiche :

marie	citroën
sophie	peugeot
	cadillac

On obtient les propriétaires de toutes les voitures (certaines voitures n'en ont pas).

1.4. Condition WHERE:

Elle exprime les critères que doivent vérifier les informations sélectionnées au moyen d'expressions booléennes.

1.4.1. opérandes: ce peut être:

-des nom de champs ,
-des constantes ,
-des noms de variables non définies dans les tables de la liste; dans ce cas SQL en demandera la valeur à l'opérateur
-une valeur non définie , dite valeur nulle ; par exemple, on pourra chercher les enfants nés de père inconnu:

SELECT prénom FROM enfant WHERE père IS NULL;

sélectionne les enfants dont le champ "père" n'a pas été rempli.

1.4.2. opérateurs de comparaisons entre champs ou entre champs et constantes:

= != > < >= <= IS LIKE

Ces opérateurs pouvant porter aussi bien sur des grandeurs numériques que sur des grandeurs alphabétiques, des dates, des clés, etc...

Nota 1: Les opérateurs <, >, etc...appliqués à des chaînes de caractères font référence à l'ordre alphabétique
ainsi une instruction telle que

SELECT nom, prénom FROM personne WHERE **nom > 'Martin';** affiche toutes les personnes dont le nom se classe après 'Martin' dans l'ordre alphabétique"

Nota 2: les comparaisons faisant intervenir durée et dates se font en convertissant tout en **jours**
exemple chercher les gens mariés depuis plus de 3 ans se fait par

SELECT * FROM personne WHERE aujoudhui( )-date_naissance >3 *** 365**

IS est utilisé en particulier pour tester la non définition d'un champ:
exemple WHERE nom IS NULL; (on n'écrit pas ...=NULL)

LIKE est utilisé pour tester une égalité approximative entre champs alphabétique, par exemple: chercher les DUPOND et DUPONT:

SELECT. . . WHERE nom LIKE 'DUPON**?**';

le caractère
? représente **un seul caractère;**
* représente **un nombre quelconque de caractères,** ainsi **LIKE 'Dupon* '** pourrait donner 'Dupond', 'Dupont', 'Dupont de Nemours'

NOTA: les comparaisons sur des champs alphabétiques sont plus longues que celles portant sur des champs numériques, d'où l'intérêt d'identifier les occurrences par des clefs numériques.

1.4.3. opérateurs booléens: AND, OR, XOR, NOT,

Exemple 5: sélectionner les clients d'un concessionnaire auto qui ont gardé leur voiture plus de 5 ans **ou** qui ne l'ont pas encore revendue:

```
SELECT DISTINCTROW achète.nom
FROM achète, vend
WHERE (achète.date_achat +5< vend.date_vente) OR NOT EXISTS
(SELECT nom  FROM  vend WHERE achète.nom=vend.nom);
```

1.4.4. opérateurs ensemblistes: UNION, IN, EXISTS

Ils permettent d'écrire plus simplement certaines conditions.

IN: appartenance à un ensemble selectionné

exemple 1: grouper des conditions d'égalité:

SELECT nom, prénom FROM personne WHERE prénom **IN** ('jean', 'pierre', 'marie');

Plus simple que:
SELECT …..WHERE (prénom= 'jean') OR (prénom= 'pierre') OR(prénom = 'marie');

L'ensemble peut être le résultat d'une sélection:

```
SELECT nom, prénom FROM personne
WHERE prénom IN  (SELECT  prénom FROM amis)
```

puisque par définition, un SELECT définit un ensemble .

On peut évidement sélectionner aussi les ennemis

```
SELECT nom, prénom FROM personne
WHERE NOT (prénom  IN  (SELECT  prénom FROM amis))
```

*attention WHERE (prénom **NOT IN** (SELECT prénom FROM amis)) est FAUX*

UNION: Exemple: chercher les personnes ayant acheté OU vendu une voiture

```
(SELECT nom FROM achète)UNION (SELECT nom FROM  vend);
```

Ceci fait l'union des deux ensembles sélectionnés

Remarque 1: l'UNION supposerait évidemment que les ensembles soient de même nature et les éléments de même format (on ne fait pas la réunion d'un ensemble de poules et d'un ensemble de lapins). Mais curieusement SQL accepte une instruction (isolée) de la forme:

```
(SELECT nom FROM achète)UNION (SELECT date_de_vente FROM  vend);
```

affichera la liste des noms puis la liste des dates

NOTA: l'union est le seul opérateur entre ensembles, l'intersection par exemple n'existe pas. Par exemple rechercher les propriétaires à la fois d'une voiture ET d'une moto se fera par:

```
SELECT nom , prénom FROM personne WHERE  (personne.nom IN
(SELECT voiture.propriétaire FROM voiture WHERE voiture.propriétaire IN
(SELECT moto.propriétaire FROM moto));
```

NOTA: autre écriture <u>beaucoup moins performante:</u>

```
SELECT nom , prénom FROM personne
WHERE  (personne.nom IN (SELECT voiture.propriétaire FROM voiture) )
AND       (personne.nom  IN (SELECT moto.propriétaire FROM moto));
```

- Dans la première solution, on construit d'abord un ensemble (limité, il y a peu de motos) de propriétaire de motos; de cet ensemble on ne garde que les propriétaires de voiture;
-dans la deuxième solution, on construit en plus un <u>très gros ensemble</u> de propriétaires de voiture pour le restreindre ensuite.

Nota: dans certains cas, l'intersection peut se réaliser par une **jointure**. exemple intersection de l'ensemble des clients et de l'ensemble des employés d'un magasin

```
SELECT *  FROM client INNER JOIN  employé  ON nom_client=nom_employé
```

Ca ne marche pas dans l'exemple des propriétaires de motos et de voitures.

EXISTS

a) Permet de tester si un ensemble est vide ou non : Exemple: rechercher les personnes qui ne possèdent pas de voiture

```
SELECT nom, prénom FROM personne WHERE NOT EXISTS
(SELECT personne.propriétaire FROM voiture WHERE voiture.propriétaire=personne.nom)
```

Remarque: il n'y a pas d'opérateur "POUR TOUT"; on le réalise grâce à EXISTS:
"pour tout X tel que C" s'écrit "il n'existe pas X tel que non C"

exemple sélectionner les personnes dont toutes les voitures sont des Fiat= sélectionner les personnes pour lesquelles il n'existe pas de voiture qui ne soit pas une Fiat

```
SELECT* FROM  personne
WHERE NOT EXISTS
        (SELECT marque  FROM personne INNER JOIN voiture ON
        voiture.n°personne=personne.n°  AND  NOT (marque='Fiat'));
```

b) permet d'exprimer des contraintes ; exemple: ajouter des personnes à une table si elle n'y est pas déjà:

```
INSERT INTO personne ( nom , prénom)
SELECT personne_tmp.nom, personne_tmp.prénom
FROM personne_tmp
WHERE NOT EXISTS
        (SELECT  nom, prénom FROM personne
        WHERE (nom=[personne_tmp].nom)
        AND (prénom=[personne_tmp].prénom));
```

1.5. Différence entre les jointures et les conditions WHERE:

Comparons les instructions sémantiquement équivalentes:

```
SELECT  enfant.prénom, parent.nom
FROM parent INNER JOIN enfant ON enfant.nom_du_père= parent.nom;
```

et

```
SELECT  enfant.prénom, parent.nom
FROM parent,  enfant WHERE enfant.nom_du_père= parent.nom;
```

a) L'expérience démontre qu'en ACCESS 2000 l'instruction utilisant le WHERE est **plus rapide.** **(ce n'est pas vrai pour les versions antérieures d'ACCESS).** On pourrait en déduire que la forme utilisant le JOIN est inutile; cependant le WHERE permet plus difficilement l'expression des "LEFT JOIN" et "RIGHT JOIN"

b) dans notre exemple, ACCESS **reconnaît dans la deuxième forme la présence d'une jointure** et exécute l'algorithme optimal. Mais ce ne serait pas vrai dans des cas bizarres; exemple:

écrire

```
SELECT client.nom, facture.montant FROM client, facture
WHERE  facture.n°client<=client.n°client  AND facture.n°client>=client.n°client
```

au lieu de

```
SELECT client.nom, facture.montant FROM client, facture
WHERE  facture.n°client=client.n°client
```

Ici ACCESS est largué et les temps de calcul peuvent devenir prohibitifs

1.6. " Extension de la notion de jointure: "théta_jointure"

Au lieu d'indiquer une égalité dans la clause ON, on peut indiquer une inégalité, exemple: réservation de chambres d'hotel: on a deux tables : "désidérata" qui comporte le nom des clients potentiels et leurs prix maximum, et une table des chambres et leurs prix. On veut proposer aux clients les solutions possibles:

```
SELECT désidérata.nom, chambres_hotel.numéro FROM
désidérata INNER JOIN chambres_hotel
ON désidérata.prix_max < chambres_hotel.prix;
```

Remarque: Ceci n'est pas réalisable en ACCESS en utilisant le mode graphique. Mais on peut l'écrire en SQL

1.7. clauses additionnelles

Les éléments sélectionnés peuvent être traités par des clauses additionnelles

1.7.1. DISTINCROW, DISTINCT

Permet de n'afficher qu'une fois des informations identiques
exemple SELECT DISTINCTROW nom, prénom FROM.... n'affichera qu'une fois les couples nom, prénom
exemple 2 SELECT DISTINCT nom, prénom..... affichera une fois seulement les couples ayant un même nom (s'il y a plusieurs prénom pour un même nom, on n'en n'affiche qu'un)

1.7.2. ORDER BY

Permet de trier les informations sélectionnées par champs; exemple:
SELECT nom, prénom FROM personne ORDER BY nom;

1.7.3. GROUP BY

Permet de regrouper les informations ayant un champ commun; Exemple:
SELECT nom, prénom FROM personne GROUP BY nom;

1.7.4. COUNT

Permet d'afficher le compte d'occurrence sélectionnées et non les occurrences elles même; Exemple:

```
SELECT   COUNT( nom) FROM personne;
```

1.7.5. SUM

Permet de faire la somme des champs; en général associé à la clause GROUP BY; Exemple:

```
SELECT nom, SUM(prix) FROM facture, personne
GROUPE BY personne.nom;
```

1.7.6. AVG (average)

Permet de calculer une moyenne; Exemple

```
SELECT nom, AVG(note) FROM examen GROUP BY nom;
```

1.7.7. MAX et MIN

Permettent d'afficher une valeur maximum (minimum)

```
SELECT nom, note FROM examen WHERE note= MAX(note);
```

1.7.8. INTO

Permet de créer une nouvelle table avec les résultats de la sélection

```
SELECT DISTINCTROW nom, prénom  AS   name, nickname  INTO  table_new
   FROM personne WHERE prénom='jean';
```

1.7.9. AS

Permet de définir un autre nom pour une table. Très utile par exemple quand on doit exprimer des relations réflexives (entre éléments d'une même table).

Exemple relation de parenté entre personne.

```
SELECT DISTINCTROW personne.nom, personne_1.mère
FROM personne INNER JOIN personne AS personne_1
ON personne.mère = personne_1.nom;
```

personne_1 est donc un "alias" de personne, cela permet de distinguer les champs d'une personne et de sa mère.

Permet également de donner un nom à un résultat intermédiaire
exemple:

```
SELECT Count(personne.nom) AS [Compte De nom] INTO effectifs FROM personne;
```

Si le compte de noms doit intervenir plusieurs fois dans des calculs, on utilise l'alias

2. Instruction de modification: UPDATE (mise à jour)

Exemple 1: modification d'état civil:

```
UPDATE  personne SET personne.nom = [nouveau nom?]
WHERE (((personne.nom)=[ancien nom ?]));
```

Exemple 2 modification des prix:

```
UPDATE voiture SET voiture.prix = 2*voiture.prix;
```

2.1. origine de la modification

La valeur d'origine peut provenir d'une saisie comme dans l'exemple 1 ci dessus.

Plus généralement, elle proviendra d'une table quelconque ;**exemple**: modification de l'état-civil d'une personne en fonction d'une table des mariages:

```
UPDATE mariages, personne SET [personne].[date de mariage] = [mariage].[date]
WHERE (([personne].[nom]=[nom époux]));
```

3. ajout: INSERT

On peut ajouter des lignes dans une table de deux manières

3.1. **ajout simple:** exemple: ajout après saisie:

```
INSERT INTO personne ( nom, prénom )
VALUES ( [quel nom?] , [quel prénom ?]) ;
```

3.2. ajout multiple: on extrait les différents enregistrements dans une autre table par un select, puis on les ajoute dans la table cible dans la table cible:
exemple: la première table est une table temporaire

```
INSERT INTO personne ( nom, prénom )
SELECT DISTINCTROW personne_tmp.nom, personne_tmp.prénom
FROM personne_tmp;
```

Nota: On pourra vérifier que les informations ajoutées n'y sont pas déjà. On utilisera la contrainte EXISTS vue au § 1.4.4.

4. suppression d'enregistrements: DELETE

4.1. suppression de tous les champs ; exemple effacement de la table temporaire:

```
DELETE personne_tmp.nom, personne_tmp.prénom
FROM personne_tmp;
```

4.2. suppression de certains champs

```
DELETE personne_tmp.nom, personne_tmp.prénom
FROM personne_tmp
WHERE ((personne_tmp.nom)=[qui supprimer ?]);
```

CHAPITRE 5
ALGEBRE RELATIONNELLE

Définitions:

relation= table (entité ou association). Une relation peut également être dynamique: ensemble d'informations résultant d'un calcul et qui n'est pas forcément rangé dans une table.

opérateur= opérations (unaires ou binaires) entre les relations .

1. opérateurs unaires

1.1. projection:

Soit R(A1,A2,...An) et E un sous ensemble A1...Ak des attributs.
Par définition, $PJ_E(R)$ est la restriction de R aux attributs de E

en SQL: $PJ_E(R)$= `SELECT A1,...Ak FROM R`

propriétés:

lemme $PJ_{E1}(PJ_{E2}(R))= PJ_{E1\cap E2}(R)$ (trivial!) on en déduit:

commutativité: $E1\cap E2=E2\cap E1$ donc $PJ_{E1}(PJ_{E2}(R))= PJ_{E2}(PJ_{E1}(R))$

associativité: $E1\cap(E2\cap E3)=(E1\cap E2)\cap E3$ donc

$$PJ_{E1}(PJ_{E2}(PJ_{E3}(R))) = PJ_{E1\cap E2}(PJ_{E2}(R))$$

ou encore

$$PJ_{E1} \text{ o } (PJ_{E2} \text{ o } PJ_{E3}) = (PJ_{E1} \text{ o } PJ_{E2}) \text{o } PJ_{E3}$$

idempotence:

$$PJ_E(PJ_E(R))= PJ_{E\cap E}(R)= PJ_E(R)$$

1.2. sélection: SL

Etant donné un critère C portant sur un ou plusieurs attributs, de la relation R on note $SL_C(R)$ la restriction de R aux lignes vérifiant C.

En SQL: $SL_C(R)$ = `SELECT * FROM R WHERE C`

propriétés:

lemme $SL_{C1}(SL_{C2}(R)) = SL_{C1 \cap C2}(R)$ (trivial!) on en déduit:

commutativité: $C1 \cap C2 = C2 \cap C1$ donc $SL_{C1}L_{C2}(R)) = SL_{C2}(SL_{C1}(R))$

associativité: $C1 \cap (C2 \cap C3) = (C1 \cap C2) \cap E3$ donc

$SL_{C1}(SL_{C2}(SL_{C3}(R))) = SL_{C1 \cap C2}(SL_{C2}(R))$ ou encore

SL_{C1} o $(SL_{C2}$ o $SL_{C3}) = (SL_{C1}$ o $SL_{C2})$o SL_{E3}

idempotence:

$SL_C(SL_C(R)) = SL_{C \cap C}(R) = SL_C(R)$

On notera la parfaite analogie entre SL et PJ qui définissent de sous ensembles respectivement de ligne et de colonnes d'une table.

Commutativité SL et PJ:

Soit PJ_E et SL_C avec un critère C qui porte exclusivement sur des attributs de E;

on a: SL o PJ = PJ o Sl

ce qui revient à dire qu'on peut supprimer d'abord des lignes (par SL) puis des colonnes (par PJ), ou l'inverse.

On a donc une seule instruction pour cette composition:

```
SELECT  A1,A2,...Ak  FROM  R  WHERE  C
```

Nota: la propriété n'a de sens que si C porte sur des colonnes non éliminées: par exemple

$PJ_{(A,B)}(SL_{(C=0)}(R(A,B,C)))$ a un sens alors que

$SL_{(C=0)}(PJ_{(A,B)}(R(A,B,C)))$ n'en n'a pas.

Ainsi: SELECT A1 FROM R WHERE A2 =0 signifie évidemment qu'on fait d'abord la sélection, puis la projection!

Critères:

toute expressions logiques sur les attributs utilisant les opérateurs de comparaison (=,<,>, etc), les opérateurs logiques AND, OR, NOT, l'opérateur IN d'appartenance à un ensemble, ou l'opérateur EXISTS d'existence d'un ensemble.

Exemple: soient les tables:

achète(nom, numéro_voiture, date_achat) et vend(nom,numéro_voiture,date_vente)

on veut sélectionner les gens qui gardent leur voiture plus de 5 ans:

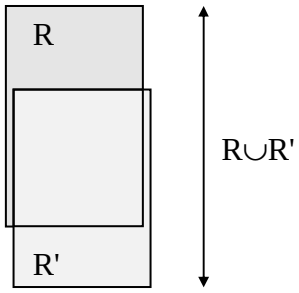
```
SELECT DISTINCTROW achète.nom
FROM achète, vend  WHERE (achète.date_achat +5< vend.date_vente) OR NOT EXISTS
(SELECT nom  FROM  vend WHERE achète.nom=vend.nom);
```

2. Opérateurs binaires ensemblistes

Etant données deux relations R(A1,. . . An) et R'(A'1, . . .,A'n) pour lesquelles les Ai et A'i ont mêmes domaine de définition, on peut définir les opérateurs UNION (∪), INTERSECTION (∩), différence (-), ou exclusif, etc... Leurs propriétés sont en tout point égales à celles sur les ensembles (les relations étant des ensembles d'occurrences).

2.1. UNION: réunion des occurrences deux relations R et R':

```
SELECT *  FROM R  UNION  SELECT * FROM R'
```



propriétés:

a) (ensemblistes) associative, commutative, idempotence.
b) commutativité avec PJ et SL (triviale)

Nota: comme on l'a vu au chapitre précédent, assez curieusement ACCESS accepte l'union de deux ensembles de nature différente; exemple:

```
SELECT nom FROM personne UNION SELECT PRIX FROM catalogue;
```

2.2. intersection: il n'y a pas d'opérateur spécifique:

cas 1: A1 est la clef:
on prend toutes les occurrences de R dont l'identifiant est également dans R':

```
SELECT  A1,A2, . . An  FROM  R WHERE  A1  IN  (SELECT A'1 FROM R' )
```

attention: peut donner des résultats idiots si par exemple, la clef est un numéro automatique dans R et dans R'.

Cas 2: pas de clef (ou clef non utilisable, ou identifiant non atomique):

```
SELECT A1,A2,…An FROM R
WHRE EXISTS  (SELECT A'1,A'2…A'n FROM R'
        WHERE (A'1=A1) and (A'2=A2) and . . .  and (A'n=An));
```

ATTENTION la séquence suivante serait fausse:

```
SELECT   A1,A2, . . . An  FROM  R WHERE
A1   IN   (SELECT A'1 FROM R' )
AND
A2   IN   (SELECT A'2 FROM R' )
AND
A3   IN   (SELECT A'3 FROM R' )
 etc..
```

Nota: exemple qui met bien en évidence le fait que d'avoir un identifiant unique simplifie bien les choses.

Propriétés (comme pour l'union):

a) (ensemblistes) associative, commutative, idempotence.

b) commutativité avec PJ et SL (triviale)

c) distributivité: $R\cap(S\cup T)=(R\cap S)\ \cup\ (R\cap T)$ et $R\cup(S\cap T)=(R\cup S)\ \cap\ (R\cup T)$

d) non distributivité de PJ pour $\cap$

exemple

R= { (a,1), (b,2)} R' ={ (a,2), (b,1)} PJ(R∩R') = PJ(vide)=**vide**

PJ(R)={a,b} PJ(R')={a,b} PJ(R) ∩PJ(R')= **{a,b}**

2.3. différence: notée R-R' (c'est plus exactement $R\cap\neg R'$): principe à peu près identique à l'intersection, on remplace IN par NOT IN:

```
SELECT A1,A2,…An FROM R
WHRE NOT EXISTS  (SELECT A'1,A'2…A'n FROM R'
        WHERE (A'1=A1) and (A'2=A2) and . . .  and (A'n=An));
```

Et non:

```
SELECT   A1,A2, . . . An  FROM  R WHERE
A1   NOT IN   (SELECT A'1 FROM R' )
AND
A2 NOT IN   (SELECT A'2 FROM R' )
AND
A3 NOT  IN   (SELECT A'3 FROM R' )
 etc..
```

2.4 ou exclusif: R ⊕ R' = (R-R') ∪(R'-R)

```
SELECT A1,A2,…An FROM R
WHRE EXISTS  (SELECT A'1,A'2…A'n FROM R'
        WHERE (A'1=A1) and (A'2=A2) and . . .  and (A'n=An))
UNION
SELECT A'1,A'2,…A'n FROM R'
        WHRE EXISTS  (SELECT A'1,A'2…A'n FROM R'
        WHERE (A'1=A1) and (A'2=A2) and . . .  and (A'n=An));
```

..

2.5. complément:

Dans la mesure ou il est en général quasi impossible de définir l'univers de toutes les occurrences relatives à un domaine, on ne peut pas définir un complément. Tout au plus peut-on définir le complément d'un ensemble dans un autre qui le contient, c'est la différence (c'est la propriété qu'on a utilisé au § 2.4.

2.6. exercices sur les opérateurs ensemblistes:

Réservation de chambres d'hotel:

étant données CHAMBRE(n°) et RESERVE(n°chambre, n°semaine),

a) écrire la requête trouvant les chambres occupées la semaine 5,
b) la requête trouvant les chambres libres la semaine 5,
c) prouver que l'union des deux requêtes donne toutes les chambres

a) $R= PR_{n°}(SL_{(n°semaine=5)}(réserve))$ =

```
SELECT chambres.n° FROM réserve
WHERE chambre.n°semaine=5;
```

b) $PR_{n°chambre}(SL_{(n°chambre\ not\ in\ R)}(chambres))$=

```
SELECT chambres.n°chambre FROM chambres WHERE  chambres.n°chambre NOT IN
(SELECT chambres.n° FROM réserve WHERE chambre.n°semaine=5)
```

c) l'ensemble des chambres est

$PR_{n°chambre}(chambres)= PR_{n°chambre}(SL_{(n°chambre\ \underline{not\ in}\ R)}(chambres))$
UNION
$PR_{n°chambre}(SL_{(n°chambre\ \underline{in}\ R)}(chambres))$

ensembles des chambres qui sont réservées la semaine 5 et de celles qui ne le sont pas.

Détaillons la sélection.

$PR_{n°chambre}(SL_{(n°chambre\ \underline{in}\ R)}(chambres))=$

```
SELECT chambres.n°chambre FROM chambres
WHERE  chambres.n°chambre  IN
(SELECT chambres.n° FROM réserve
 WHERE chambre.n°semaine=5])
```

or, en admettant qu'on n'a pu réserver que des chambres existantes, on a :

$PR_{n°}$ (réserve) $\subset PR_{n°chambre}$(chambres)

Donc la sélection supplémentaire $PR_{n°chambre}(...)$ n'ajoute rien (puisque ce qu'on sélectionne dans "réserve" est automatiquement dans "chambres"); on a donc

$PR_{n°chambre}(SL_{(n°chambre\ \underline{in}\ R)}(chambres))= R$

$= PR_{n°}(SL_{(n°semaine=5)}($ réserve))

on a donc bien:

$PR_{n°chambre}$(chambres)=

$PR_{n°chambre}(SL_{(n°chambre\ not\ in\ R)}$(chambres)) UNION $PR_{n°}(SL_{(n°semaine=5)}($ réserve))

CQFD

3. Produit cartésien et jointures.

3.1. Produit cartésien PC:

On appelle **produit cartésien** de deux relations quelconques R et S dont les occurrences sont Ri et Sj la relation T formée de tous les couples {Ri,Sj }

R PC S= | SELECT * FROM R,S |

Nota:

Il se peut que R et S aient des champs d'un même domaine par exemple R(A,B) et S(A,C). on fait en fait le produit cartésien de l'ensemble des {Ai,Bi} avec l'ensemble des{Aj,Cj}, on obtient des quadruplets {Ai,Bi,Aj,Cj}, mais on convient de regrouper les Ai et Aj . Exemple:

R= motorisation{voiture, moteur}, S=carosserie{voiture,couleur}

Le produit cartésien est par définition formé de tous les triplets {voiture,moteur,couleur} et non des quadruplets {voiture, voiture, moteur, couleur}.

Propriétés:

le produit cartésien est **associatif, commutatif** (à un réordonnancement près des attributs),

distributivité à gauche et à droite pour l'union: A PC (B∪C) = (A PC B) ∪ (A PC C)

le deuxième membre est en effet formé de tous les {Ai,Bj} et de tous les {Ai, Ck}, donc de tous les {Ai, Dh} avec Dh = Bi ou Cj, donc D=B∪C.

distributivité à gauche et à droite pour l'intersection:

A PC (B∩C)= (A PC B) ∩ (A PC C)

En posant D= B∪C, le premier membre est en effet formé de tous les {Ai, Dh} ou Dh est un Bi et un Ck; donc on a les {Ai,Bj} qui sont aussi des {Ai,Ck}

En général non distrbutivité pour SL

exemple

$SL_{(A=C)}$(R(A,B) PC S(B,C)) n'est pas la même chose que $SL_{(A=C)}$R(A,B) PC $SL_{(A=C)}$ S(B,C)): en effet $SL_{(A=C)}$R(A,B) n'a pas de sens.

Cas particulier: si la sélection ne porte que sur les champs d'une même relation, il y a trivialement distributivité. Exemple:

$SL_{(A>1000)}$(R(A,B) PC S(B,C)) = $SL_{(A>1000)}$R(A,B) PC $SL_{(A>1000)}$ S(B,C)) à condition de convenir que $SL_{(A>1000)}$ S(B,C)=S(B,C), la contrainte étant inopérante.

Relation avec les projections:

Théorème

Soient R1(A,B) et R2(P,Q) où les champs A,B,P,Q sont des ensembles de champs indépendants, **le produit catésien de deux projections est une projection du produit cartésien:**

$$(PJ_A\,(R1(A,B))\ )\ PC\ (\ PJ_P\,R2(P,Q))) = PJ_{A,P}(R1(A,B)\ PC\ (\ R2(P,Q))$$

en effet:

a) dans la partie gauche:

R1 est formé de couples {Ai,Bi}; PJ_A (R1(A,B)) est donc formé de tous les Ai distincts . De même R2 est formé de couples {Pj,Qj}; PJ_P (R2(P,Q)) est donc formé de tous les Pj distincts. Le produit cartésien du premier membre est formé de tous les couples possibles {Ai,Pj} distincts.

b) dans la partie droite:

le produit cartésien R1 PC R2 est formé de tous les quadruplets {Ai,Bi,Pj,Qj}; sa projection sur A,P donne tous les couples {Ai,Pj} distincts

C.Q.F.D.

Nota: l'indépendance de A,B,P,Q est essentielle: Soit la relation R(A,B,C), et ses projections R(A,B)= $PR_{(A,B)}$(R) et R(A,C)= $PR_{(A,C)}$(R). Ces deux relations ne sont pas indépendantes, ayant A en commun. On n'a pas R(A,B) PC R(A,C) = R(A,B,C): le produit cartésien des projections d'une même relation ne redonne pas la relation initiale:

Exemple: modèles disponibles de voitures= R(voiture,moteur, couleur)=

Voiture	moteur	couleur
306	essence	verte
306	essence	bleue
306	diesel	verte
605	essence	rouge
605	diesel	bleue

R(voiture,moteur)=

Voiture	moteur
306	essence
306	diesel
605	essence
605	diesel

R(voiture,couleur)=

voiture	couleur
306	verte
306	bleue
605	verte
605	rouge

produit cartésien:

Voiture	moteur	couleur
306	essence	verte
306	essence	bleue
306	diesel	verte
306	*diesel*	*bleue*
605	essence	rouge
605	*essence*	*bleue*
605	*diesel*	*rouge*
605	diesel	bleue

le produit cartésien des deux projections n'est pas une projection , il y a même des termes en trop: les modèles en italique ne sont pas disponibles!

Nota: le mal vient de la projection qui fait perdre de l'information: par exemple on perd l'information "la 306 diesel n'est pas disponible en bleue". Curieusement, perdre de l'information fera apparaître des occurrences supplémentaires!
On peut dire aussi que le mal vient de la fusion conventionnelle des champs communs aux deux relations du produit: dans l'opération, on fusionne les deux champs "voiture"; mais la 306 diesel n'est pas la même 306 que la 306 bleue!

Cas particulier:

$R(A,B)$ PC $(PJ_P(S(P,Q)) = PJ_P(R(A,B)$ PC $S(P,Q))$ (éliminer les champs indésirables avant ou après le produit cartésien ne change rien sur le résultat; par contre pour des raisons d'efficacité, il vaudra toujours mieux faire les produits cartésiens sur des ensembles les plus petits possibles, donc faire la projection en premier).

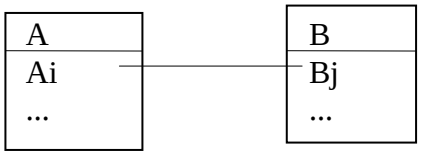
Par contre on a bien
$PR_{(A,B)}(R(A,B)$ PC $R(A,C)) = R(A,B)$ et $PR_{(A,C)}(R(A,B)$ PC $R(A,C)) = R(A,C)$:
les projections du produit cartésien redonnent les relations initiales.

3.2. Jointures.

Une jointure est un sous ensemble du produit cartésien défini par l'égalité de deux colonnes de même domaine.

3.2.1. Jointure naturelle:

représentation graphique:

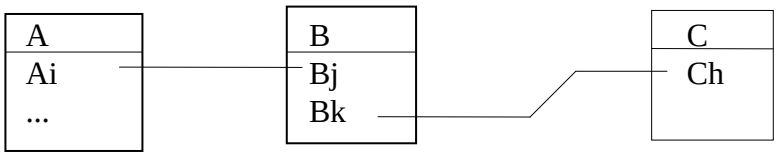


en SQL:

SL $_{Ai=Bj}$(A PC B) = SELECT * FROM A INNER JOIN B ON A.Ai=B.Bj

ici le mot INNER (intérieur) signifie qu'aucun des Ai ou Bj n'est "nul" (par opposition au "LEFT JOIN" ou "RIGHT JOIN" ci dessous).

Nota: on peut avoir des jointures de jointures



en SQL:

```
SL (Ai=Bj)∩(Bk=Ch)((A PC B) PC C)=
        SELECT * FROM  (A  INNER JOIN  B ON A.Ai=B.Bj)
        INNER JOIN C ON B.Bk=C.Ch
```

Nota:

a) pas de limitation au nombre de jointures successives sinon la complexité effarante de l'instruction SQL.
b) la jointure peut se faire à l'intérieur de la relation, par exemple pour trouver le père ou le grand père de quelqu'un; exemple:

```
SELECT DISTINCTROW personne.nom, personne_1.nom
FROM personne INNER JOIN personne AS personne_1 ON personne.mère =
personne_1.nom WHERE personne.nom="dupont";
```

autre forme SELECT * FROM A,B WHERE A.Ai=B.Bj
ce qui montre qu'une jointure est une sélection dans le produit cartésien

<u>Propriétés:</u> en partie déduites de celle du produit cartésien

a) on n'a pas R(A,B) JN R(A,C) = R(A,B,C): la jointure des projection ne redonne pas la relation initiale.

b) Par contre on a bien
$PR_{(A,B)}(R(A,B)\ JN\ R(A,C)) = R(A,B)$ et $PR_{(A,C)}(R(A,B)\ PC\ R(A,C)) = R(A,C)$: les projections du produit cartésien redonnent les relations initiales

c) commutativité PJ/PC:
soient R(A,B) et S(C,D); la projection du produit cartésien est égale au produit cartésien des projections:

corollaire: exercice des chambres d'hotel

étant données CHAMBRE(n°) et RESERVE(n°chambre, n°semaine),
on crée la requête en conservant la jointure chambre.n°=réserve.n°chambre:

SELECT chambres.n° FROM réserve INNER JOIN chambres ON
chambre.n°=réserve.n°chambre WHERE chambre.n°semaine=5;

ou encore
$PJ_{(n°)}(SL_{(C1)}$(Réserve PC chambre)) =
SELECT chambres.n° FROM réserve, chambres WHERE
(chambre.n°=réserve.n°chambre) AND (chambre.n°semaine=5);

comparons avec
$PJ_{(n°)}(SL_{(C)}$(Réserve PC chambre)) = SELECT chambres.n° FROM réserve, chambres WHERE (chambre.n°semaine=5);

on constate que le critère C1 est plus strict que C et donc qu'on obtiendra <u>moins</u> de chambres; en particulier on n'aura pas:

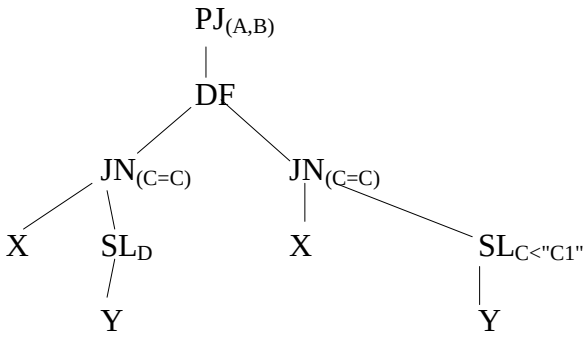
$PJ_{(n°)}(SL_{(C1)}$(Réserve PC chambre)) UNION $PR_{n°}(\ SL_{(n°semaine=5)}$(réserve)) = chambre

(il manquera toutes les chambres qui n'ont jamais été réservées)

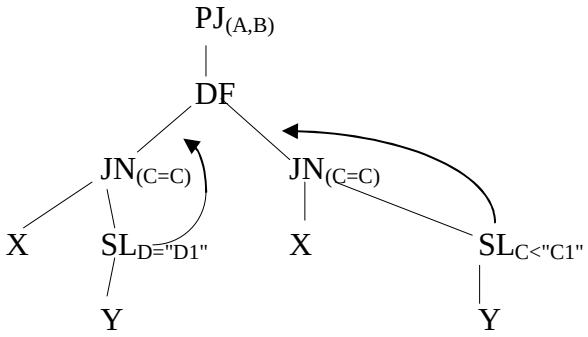
Autre exemple des propriétés algébriques: étant données personne(nom, prénom, fonction) et statut(fonction, salaire), on cherche les ingénieurs dont le salaire est inférieur à 100KF:
Solution non optimale

$PJ_{(nom,prénom)}$ DF((personne $JN_{(fonction)}$ $SL_{fonction="ingénieur"}$ statut)
,(personne $JN_{(fonction)}$ $SL_{salaire<100}$ Statut))

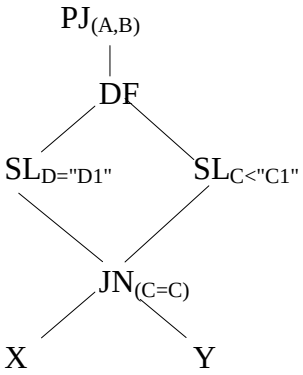
pour simplifier appelons X(A,B,C) et Y(C,D) ces relations et considérons la formule
$PJ_{(A,B)}$ DF((X $JN_{(C=C)}$ $SL_{D="D1"}$ Y) , (X $JN_{(C=C)}$ $SL_{C<"C1"}$ Y))
représentons cette formule sous forme d'arbre:



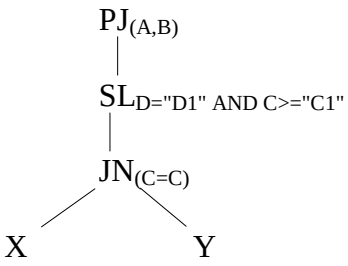
On remarque que les deux branches du bas ne diffèrent que par les sélections ➔ on fait des permutations suivantes:



on obtient:



Or $(SL_{C1}(R)) - (SL_{C2}(R)) = (SL_{(C1 \text{ AND NOT } C2)}(R))$ on simplifie donc encore en:



et pour aller plus vite on repermute SL et JN pour obtenir

$PJ_{(A,B)}$ $((SL_{D="D1" \text{ AND } C>="C1"}(X)) JN (SL_{D="D1" \text{ AND } C>="C1"}(Y)))$
on s'apperçoit que la sélection sur X n'apporte rien

ou encore

$PJ_{(nom,prénom)}$(personne JN $(SL_{(fonction=ingénieur \text{ AND } salaire<100)}$ (statut)))

exercice 2: base de donnée d'un hopital
soient les relations: docteur(n°, nom, département), patient(n°, nom, N°docteur), soin(N°patient, médicament, quantité).

On cherche qui a administré du viagra à des bébés:

1 solution (**idiote**)

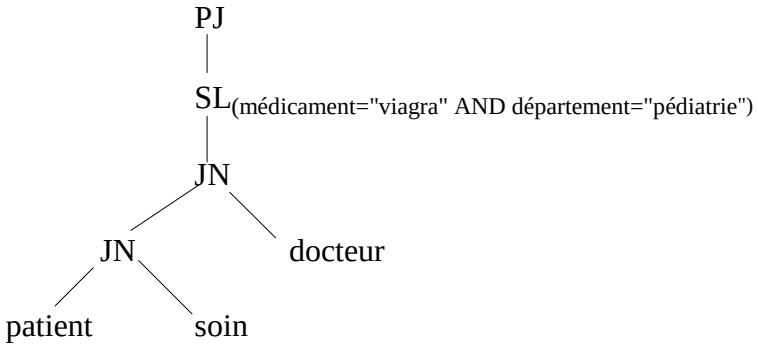
```
R=SELECT docteur.nom, patient.nom FROM
    patient , soin, docteur
   WHERE (soin.n°patient=patient.n°) AND
   (Docteur.n°=patient.n°docteur) AND
   (médicament="viagra" ) AND (département="pédiatrie")
```

nota: ordre de grandeur : 100 médecins, 10000 patients, 1000 soins possibles➔10 milliards de combinaisons à créer avant de faire les tests➔nécessité d'optimiser!

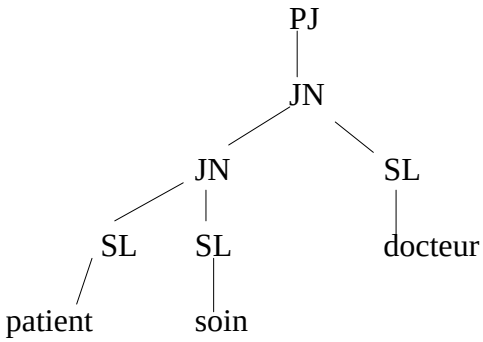
2 solution pas trop idiote:
SELECT docteur.nom, patient.nom FROM
((patient INNER JOIN soin ON soin.n°patient=patient.n°)
INNER JOIN docteur ON Docteur.n°=patient.n°docteur)
WHERE (médicament="viagra" AND département="pédiatrie")

$PJ_{(docteur.nom,patient.nom)}(SL_{(médicament="viagra"AND département="pédiatrie")}$
((patient JN soin)JN docteur)))

arborescence:



remarque 1: la sélection des jointures est la jointure des sélections donc on arrive à



en limitant les selections aux champs présents dans les tables. On effectue donc des jointures sur des tables beaucoup plus petites.

Remarque 2: On remarque en outre qu'il est inutile de faire des jointures complètes: si des champs ne doivent pas servir "plus haut", on peut les supprimer ➔ donc faire des projections préalables.

forme définitive:

$PJ_{(docteur.nom,patient.nom)}((PJ_{(n°, nom)}$ patient JN $SL_{(médicament="viagra)}$ $PJ_{(N°patient, médicament)}$ soin) JN $SL_{(département="pédiatrie")}$ docteur)

remarque 3: ordre des jointures
vaut-il mieux faire (patient, soin)docteur ou (patient, docteur)soin ou (docteur, soin)patient

de toutes évidence le troisième: il n'y aura peut-être qu'un docteur assez con pour ce traitement, il suffira de chercher qui il a traité, alors qu'il y a beaucoup de couples (patient, soin) et (patient, docteur).

NOTA: réalisation en ACCESS:

Access n'accepte pas de faire des jointures entre des sélections du type

SELECT ….FROM (SELECT…FROM…) INNER JOIN (SELECT…)

Par contre on peut passer par deux requêtes intermédiaires:
SELECT ….FROM R1 INNER JOIN R2

CHAPITRE 6
Normalisation des bases de données

Principes de base

La normalisation est l'application d'un certain nombre de règles destinées à améliorer les performances de la base, aussi bien en encombrement qu'en temps d'exécution des requêtes. Ces règles ont été hiérarchisées en plusieurs niveaux allant de la 1ère forme normale à la sixième.

Chaque forme normale de niveau i est une forme normale de niveau i-1 à laquelle on a rajouté des contraintes supplémentaires.

L'idée générale est qu'il faut décomposer ce qui est compliqué en éléments simples; c'est le proverbe **"diviser pour règner".** les relations complexes seront décomposées en éléments plus simples dont on fera la jointure.

8.1. Un peu de théorie (ce paragraphe peut être sauté dans un premier temps)

8.1.1. dépendance fonctionnelle

Une DF est une dépendance entre un ensemble A={A1,A2,...An} de champs "sources" et un ensemble B={B1,B2,. . .,Bk} de champs "résultats". la connaissance de A implique la connaissance de B. on note ceci $a \rightarrow b$ *avec la convention:*

a = "on connaît A", b="on connaît B"

Cette opération "implication" est très classique en logique et en algèbre de Boole.; elle peut s'écrire dans cette algèbre:

$a \rightarrow b = \overline{a} + b$ *ce qui peut s'interpréter comme "ou bien a est faux ou bien b est vrai".*

Ainsi une dépendance fonctionnelle telle que {A1, A2, A3} définit B s'exprime par l'expression Booléenne:

$$\overline{a1.a2.a3} + b = \overline{a1} + \overline{a2} + \overline{a3} + b$$ *(ou un des Ai est inconnu ou B est connu).*

De même A définit {B1, B2, B3, B4} s'exprime par

$\overline{a} + b1.b2.b3.b4$ *(ou A est inconnu ou tous les Bi sont connus)*

8.1.2. plusieurs dépendances fonctionnelles

Quand on dit que A détermine B et C détermine D, il' s'agit d'un "et logique"; on traduit ceci par :

$A \rightarrow B$ *et* $C \rightarrow D = (\overline{a} + b) . (\overline{c} + d)$

Ceci se généralisant à un nombre quelconque de dépendances fonctionnelles. Donc l'ensemble des DF d'une base de données se caractérise par une expression booléenne en produit de sommes .

8.1.3.Un rappel d'algèbre de Boole: la dualité.

En algèbre de Boole, les opérateurs + et x ont exactement les mêmes propriétés (l'un est le minimum, l'autre le maximum pour la relation 0 < 1) .

Ainsi traiter une expression telle que ab+cd fait appel exactement aux mêmes mécanisme que traiter l'expression (a+b)(c+d); on peut par exemple développer:

(a+b)(c+d) = ac+ad+bc+bd

donc de la même manière:

ab+cd = (a+c)(a+d)(b+c)(b+d)

Application: *simplification des expressions*

Exemple 1: *considérons les deux dépendances fonctionnelles*

{A1,A2} ➔B et (A1) ➔B

en algèbre de Boole on écrirait:

$(\overline{a1} + \overline{a2} + b)\ (\overline{a1} + b)$

expression non minimale qu'on peut simplifier, par exemple en remarquant sous forme duale que:

$\overline{a1} \,.\, \overline{a2} \,.\, b + \overline{a1} \,.\, b = \overline{a1} \,.\, b$ *(suppression des multiples)*

donc le terme $\overline{a1} \,.\, \overline{a2} \,.\, b$ *correspondant à la DF {A1,A2} ➔B, est inutile*

Exemple 2*: considérons par exemple l'expression des définitions fonctionnelles:*

(A ➔B)(B ➔C)(A ➔C)

qui s'écrit en algèbre de Boole

$(\overline{a} + b)(\overline{b} + c)(\overline{a} + c)$

expression duale de

$\overline{a}\, b + \overline{b}\, c + \overline{a}\, c$

tout étudiant de première année sait simplifier cette expression, en constatant (par exemple grâce à un diagramme de Karnaught ou par consensus) que le terme $\overline{a}\, c$ *est inutile:*

$\overline{a}\, b + \overline{b}\, c + \overline{a}\, c = \overline{a}\, b + \overline{b}\, c$

*On retrouver donc par un **calcul formel** ce qu'on aurait pu deviner :*

si connaissant A j'en déduit B et si connaissant B j'en déduit C,
alors connaissant A j'en déduit C

***nota:** ce qui est évident ici n'est pas toujours simple: essayez de simplifier l'ensemble des dépendances fonctionnelles suivant:*

(A ➔B)(B ➔A)(B ➔C)(C ➔B)

En utilisant un simple diagramme de Karnaught, il est facile de montrer que ceci peut se réduire à:

(A ➔B)(B ➔C)(C ➔B) *ou à:* *(A ➔C)(C ➔B)(B ➔C)*

mais là, le raisonnement intuitif ne suffit pas !

moralité: révisez votre algèbre de Boole

8.1. Première forme normale: 1FN:

Règle 1: chaque entité a un identifiant unique;

Exemple1

Client{nom, prénom, type_client, secteur} **n'est pas** 1FN: **il n'y a pas** d'identifiant (risque d'homonymies).

Exemple 2

Produit{code_produit, désignation, fabriquant, prix} **n'est pas** 1FN, il y a **deux** identifiants, le code et la désignation du produit .(C'est mauvais car il y a risque d'incohérence entre les deux.

Exemple 3

Client{N°INSEE, nom, prénom, …) **est 1 FN** car le n°INSEE est l'identifiant unique.

Règle 2 tous les attributs sont atomiques.

Exemple 1: L'attribut "date" **n'est pas atomique**, il peut se décomposer en "jour", mois", "année". L'intérêt de cette décomposition sera par exemple de faire un mailing à tous les clients nés un 13 mai pour leur anniversaire, ce qui serait difficile sinon impossible si la date est une chaîne de caractère sans délimiteurs.

Exemple 2 l'attribut "adresse" n'est également pas atomique (il se décompose en N°, rue, ville, département, etc…) l'intérêt est de pouvoir isoler les habitants d'une ville donnée, d'une rue donnée.

Nota: cette notion d'atomique doit être appréciée en termes d'utilisation dans la base: si par exemple on est certain que la date n'aura jamais à être décomposée, on dira qu'elle est atomique.

De la même manière, tous les attributs doivent être des scalaires, pas des tableaux (de toute manière c'est impossible en ACCESS).

8.2. Deuxième forme normale 2FN

voir "dépendances fonctionnelles"

2FN= 1FN + Règle:"toutes les dépendances fonctionnelles sont élémentaires".

Une DF {a1,a2,„..ak,…an}➔ y est dit élémentaire s'il n'existe pas {a1,a2,„..ak}➔ y

Exemple 1

Dans la base de données d'un hôpital, une entité **patient** a pour clef primaire le N° de sécurité sociale (celui du chef de famille) et le prénom du patient (enfants par exemple) et pour champ le téléphone

Patient
#N° sécurité sociale
nom
#prénom
téléphone

il y a une dépendance fonctionnelle (N°, prénom)➔ téléphone mais également la dépendance fonctionnelle plus simple (N°)➔ téléphone (si tous les membres de la même famille on le même téléphone).

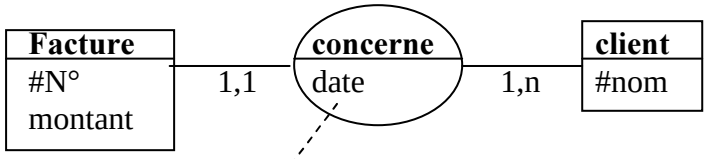
Pour normaliser, il faudra décomposer la table en deux

Patient
#N° securité sociale
nom
#prénom

ET

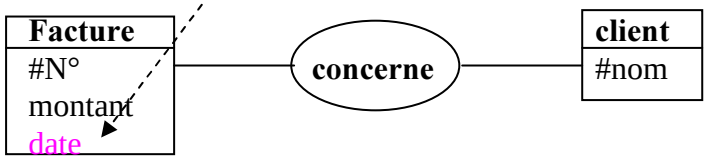
téléphone
#N° securité sociale
téléphone

Exemple 2:



La définition fonctionnelle { N°, nom } ➔ date n'est pas élémentaire puisque N°➔date.

Pour normaliser, il suffira de déplacer la date:



NOTA: d'une manière formelle, la théorie peut se traiter en faisant un simple calcul booléen (CF §8.1.3.) en constatant que :

{a1,a2,…ak,..an}➔ y et {a1,a2,..ak}➔ y = {a1,a2,,..ak}➔ y

puisque

($\overline{a1}$ + $\overline{a2}$+…$\overline{ak}$ + …+ $\overline{an}$ + y)($\overline{a1}$ + $\overline{a2}$+…$\overline{ak}$ + y) = $\overline{a1}$ + $\overline{a2}$+…$\overline{ak}$ + y

autrement dit

toute DF qui n'est pas élémentaire peut être éliminée.

Donc éliminons la !

8.3. troisième forme normale 3FN

Règle: 3FN= 2FN+ règle: toutes les DF sont directes

Une Dépendance fonctionnelle A➔ B est dite directe s'il n'existe pas C tel que A➔C et C➔B.

Autrement dit il n'y a pas de pas de "raccourcis"

Exemple 1:

Personne
#nom
nombre d'heures mensuelles
taux horaire
montant salaire

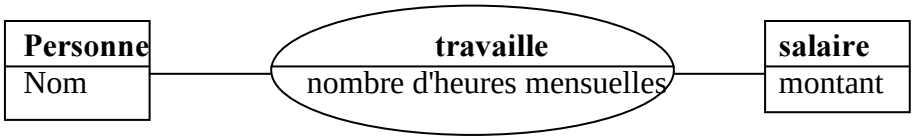
on observe 3 dépendances fonctionnelles:

nom ➔montant_salaire

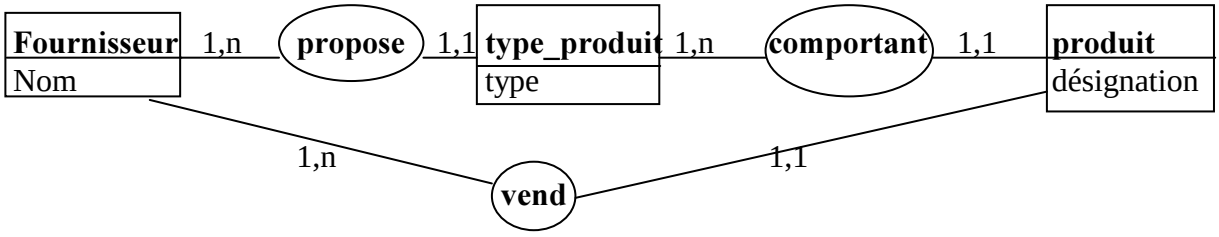
nom ➔ (nombre d'heures mensuelles, taux horaire)

(nombre d'heures mensuelles, taux horaire)➔ montant_salaire

Ce n'est pas 3FN. Pour normaliser, il suffit de supprimer le champ salaire qui peut être calculé à la demande. On peut aussi introduire une entité et une association nouvelle :



Exemple 2

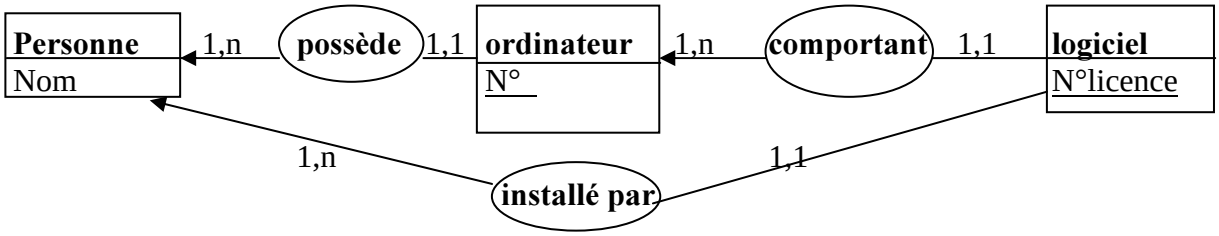


Ce MCD n'est pas 3FN puisque
Désignation ➔ type ➔nom **et** désignation➔nom. Cette dernière DF n'a pas d'utilité et l'association "vend" peut être supprimée.

Nota: ceci peut concerner un nombre quelconques de DF en raison de la transitivité**, il ne doit pas exister A➔B, B➔C, C➔D, D➔E et A➔F, F➔E**

Aspects formels:
Comme il a été vu au paragraphe 8.1.3. l'ensemble des DF de la base peut être décrit par une expression booléenne, et simplifié par des méthodes classiques de l'algèbre de Boole. Plus précisément, simplifier A ➔C quand on a A ➔B et B ➔C revient à appliquer la théorie des consensus pour la minimisation des fonctions booléennes.

Attention des cas bizarres mais néanmoins corrects peuvent se présenter, avec pourtant la même structure: exemple



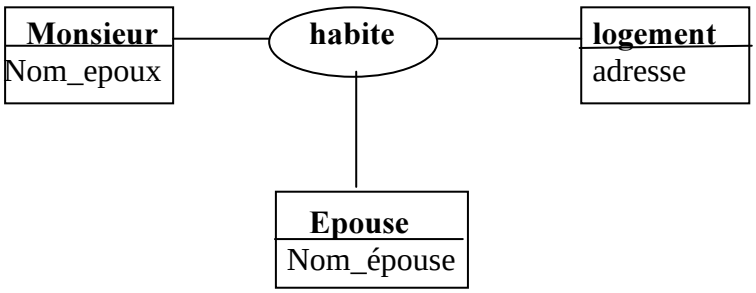
le chemin N°licence➔type➔personne et le chemin direct N°licence➔personne ne donnent pas la même personne ; donc les deux chemins sont indipensables.

Ici on a groupé abusivement deux types de personnes: "personne propriétaire" et "personne_installateur". Il n'y a pas possibilité de supprimer l'association "installé par"
Pour normaliser, il faudra décomposer la table personne (avec héritage éventuel)

8.5. Quatrième forme normale: projection des dépendances multivaluées

voir également ci dessus au § projections

8.5.2. Exemple 1:



Considérons une collection telle que:

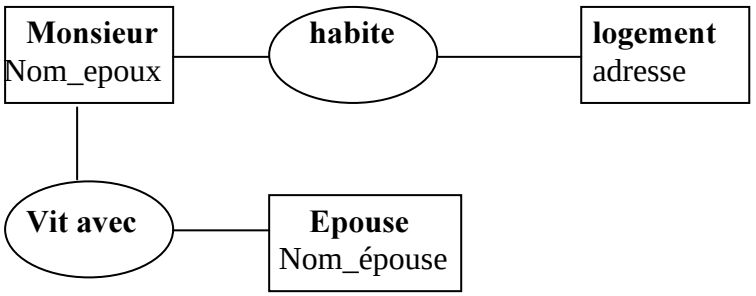
Monsieur	Madame	Adresse
Jean dupont	sophie dupont	paris
Alain martin	marie martin	bordeaux
Albert durant	jeanne dupont	amiens

On peut manifestement projeter R="habite" en deux associations R1= "vit avec" et R2="habite" dont les collections seront respectivement:

Jean dupont	sophie dupont
Alain martin	marie martin
Albert durand	jeanne dupont

jean dupont	paris
alain martin	bordeaux
albert durand	amiens

On n'a pas perdu d'information.
On peut dire sans se tromper que si X vit avec Y et X habite Z est équivalent à X habite Z avec Y . La bonne structure est donc:

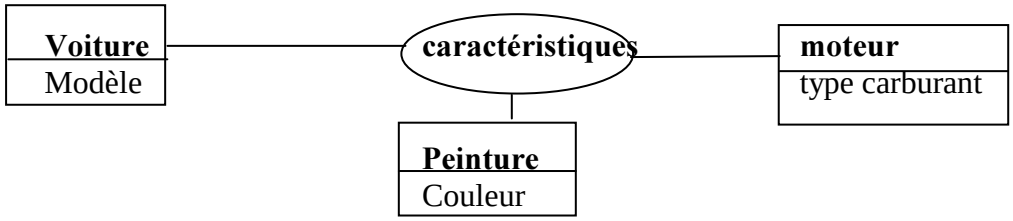


Remarque on aurait aussi bien pu décomposer en habite_1(monsieur, logement) et habite_2(madame, logement), s'ils habitent le même logement, ils vivent ensemble.

8.5.3. Question: peut on toujours projeter? Réponse **NON**:

Il y a deux raisons à cela:

8.5.3.1. Raison 1: toutes les associations ne sont pas projetables. considérons le modèle:



La collection de "caractéristiques" étant:

Modèle	moteur	Couleur
306	diesel	Bleue
306	diesel	Verte
306	essence	Bleue
306	essence	Verte
605	essence	rouge
806	diesel	blanche
806	gpl	bleue

modèle	couleur
306	verte
306	bleue
605	rouge
806	blanche
806	bleue

=R1

R2=

modèle	moteur
306	diesel
306	essence
605	essence
806	diesel
806	gpl

Interprétation : R1 JN $_{\text{modèle}}$ R2

modèle	moteur	couleur
306	bleue	diesel
306	verte	diesel
306	bleue	essence
306	verte	essence
605	rouge	essence
806	***Blanche***	***Gpl***
806	***bleue***	***diesel***
806	blanche	diesel
806	Bleue	Gpl

Erreur !

On constate que les informations déduites des deux projections sont incorrectes
Remarque, on peut vérifier que les deux autres projections, sur {modèle, couleur} et {couleur, moteur} ainsi que sur {modèle, moteur) et {couleur, moteur} sont est également incorrectes. On traitera plus loin du cas où les trois projections sont nécessaires.

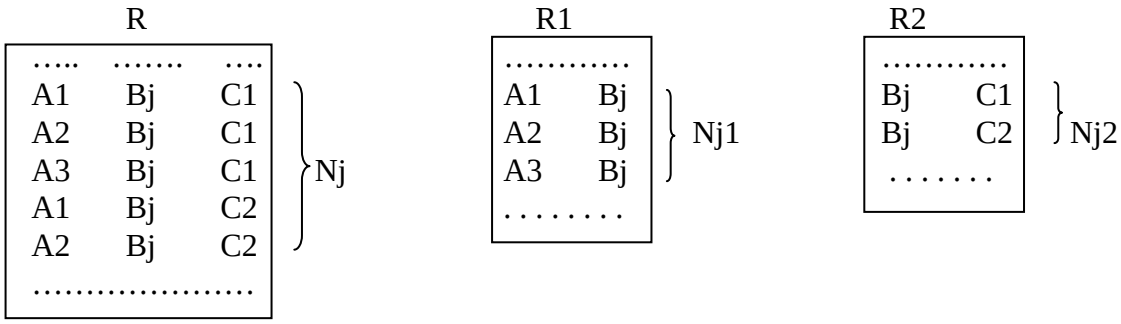
Définition:

Une association R telle que les projections R1 et R2 redonnent par composition R est appelée une dépendance **"multivaluée**" (DM)

Analyse du problème

Soit une relation R{A,B,C} décomposée (à tort ou à raison) en R1(A,B) et R2(B,C).L'ensemble des deux relations est interprété comme un jointure sur B entre les relations R1 et R2: pour chaque Bj, on considère toutes les combinaisons des Ai et Ck correspondants pour former des triplets{Ai, Bj, Ck}

en effet soit Nj, Nj1, Nj2 le nombre de lignes d'une valeur Bj du champ B de respectivement R, R1, R2.



Quand on fait la jointure, {R1 JN_B R} on combine toutes les occurrences des Ai et Ck correspondants à un même Bj; on obtient ainsi Nj1.Nj2 triplets (ici 6 triplets, alors que la relation de base n'en a que 5). Si (Nj1.Nj2) ≠ Nj pour un j donné il y a une erreur.

Ici j=5, nombre premier qui ne sera jamais un produit

Dans l'exemple précédent, il y a deux lignes "806" dans R, mais deux lignes 806 dans R1 et dans R2 donc dans la jointure, on créera 4 lines 806.
Par contre, pour les 306, il y a 4 lignes dans R, 2 dans R1, 2 dans R2, la jointure sera correcte pour les 306.

On peut alors faire les remarques de bon sens suivantes

a) Si la jointure se fait sur un champ qui est l'identifiant pour une des relations, (par exemple pour R1: ce sera certainement décomposable, car alors **Nj1=1.**

exemple R={N°INSEE, nom, prénom, n°rue, rue, ville} décomposé en
R1={N°INSEE, nom, prénom) et R2={N°INSEE, n°rue, rue, ville}

Dans R il y a un nombre quelconque de lignes pour chaque N°INSEE (une même personne pouvant avoir plusieurs adresses). Mais dans R1, le N°INSEE est la clef donc Nk1=1 pour toutes les personnes k. On est alors certain que $\forall j,\ Nj1.Nj2=Nj$, la dépendance est multivaluée, donc R est bien décomposable.

b) Si on peut affirmer que pour chaque Bj, toutes les combinaisons de Ai et Ck sont possibles (par exemple si on peut dire que toutes les peintures de voitures sont disponibles dans tous les modèles quelque soit la motorisation,) alors c'est certainement décomposable.

Remarque
Le choix de la décomposition et du champ pris pour réaliser la jointure est important.
Exemple soit R décomposé en:

a	p	x
a	p	y
b	p	x
b	p	y
c	q	x
c	q	y

⇨

R1=

a	p
b	p
c	q

R2=

a	x
a	y
b	x
b	y
c	x
c	y

on a R1 JN R =R

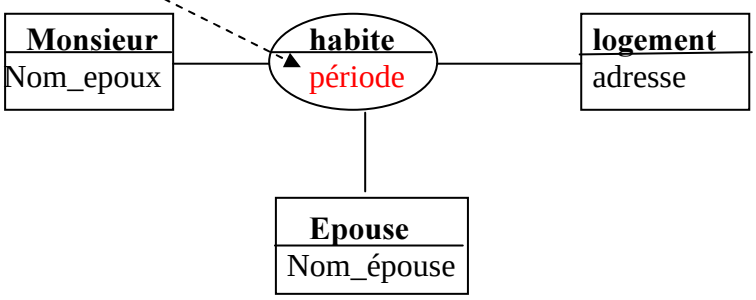
R3=

p	x
p	y
q	x
q	y

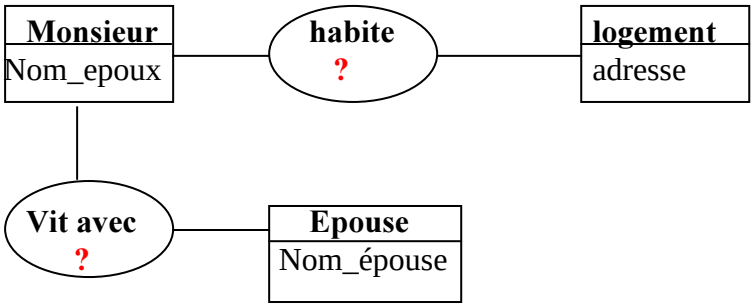
on n'a pas R2 JN R3 = R
par exemple la ligne nouvelle
{a q x}apparaît.

8.5.3.2. Raison 2: certaines associations comportent, outre les clefs externes, des champs propres

Exemple:



L'information "période" est fonction du triplet {nom_époux, nom_épouse, logement}. Il y a ici une **DF**, dépendance fonctionnelle {nom_époux, nom_épouse, logement}➔ période Si on décompose comme précédemment en:



- On ne peut mettre la date dans "habite" : monsieur peut avoir habité son logement avant son mariage avec madame
- On ne peut mettre la date dans "vit avec": monsieur peut vivre avec madame dans des logements différents à des dates différentes.

Dans un tel cas, (où existe une DF) la décomposition changerait la nature des informations.

CONSEQUENCE

On peut décomposer une association si c'est une DM et s'il n'y a pas de DF

Définition

Normalisation 4 FN = 3FN + toutes les associations vérifient "il n'y a pas de DM ou il existe une DF"

En bref, c'est 4FN **si tout ce qui est décomposable est décomposé**

Remarque: ce qui a été vu pour des associations à 3 entités se généralise à un nombre quelconque d'entités

8.5.4. Pérennité de la décomposition.

Une question fondamentale est la raison de la multivaluation. Trois cas se présentent:

8.5.4.1. on peut **démontrer** que l'association est multivaluée.
Exemple ci dessus:
si **X vit avec Y** et **X habite Z** est équivalent à: **X habite Z avec Y**

6.5.4.2. On constate (à un instant donné) que les données sont telles que la dépendance est multivaluée. Exemple:

Modèle	moteur	Couleur
306	diesel	Bleue
306	diesel	Verte
306	essence	Bleue
306	essence	Verte
605	essence	rouge

Projections

modèle	couleur
306	verte
306	bleue
605	rouge

modèle	moteur
306	diesel
306	essence
605	essence

interprétation

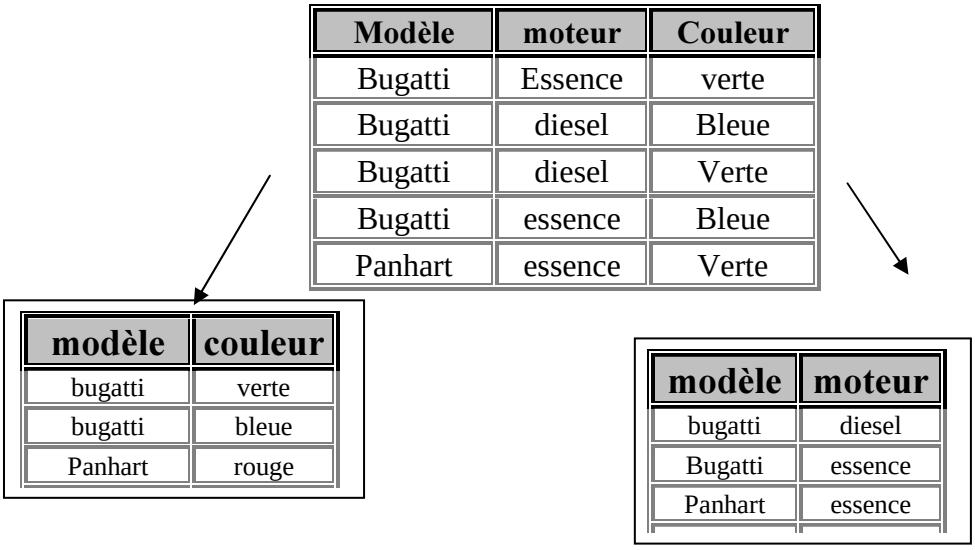
modèle	moteur	couleur
306	bleue	diesel
306	verte	diesel
306	bleue	essence
306	verte	essence
605	rouge	essence

On a constaté qu'**actuellement** c'est décomposable**. Mais il suffit que le constructeur ajoute ou supprime une ligne dans les caractéristiques des 306 pour que ce ne soit plus vrai!**

Corollaire: une association temporairement multivaluée ne doit pas être décomposée.

Nota: en reprenant la règle de décomposabilté vue ci dessus (n=n1.n2 pour chaque champ réalisant la jointure, on peut affirmer que si n=n1.n2 à l'instant t et qu'on peut faire varier n de 1, on n'aura certainement pas n+1= (n1+1).n2 ni n+1= n1.(n2+1)

8.5.4.3 on constate que la dépendance est multivaluée, et **on sait que la base ne variera jamais**
par exemple si au lieu de modèles actuels on avait des modèles de **voitures disparues** (musée de l'auto par exemple)



Modèle	moteur	Couleur
Bugatti	Essence	verte
Bugatti	diesel	Bleue
Bugatti	diesel	Verte
Bugatti	essence	Bleue
Panhart	essence	Verte

modèle	couleur
bugatti	verte
bugatti	bleue
Panhart	rouge

modèle	moteur
bugatti	diesel
Bugatti	essence
Panhart	essence

C'est multivalué et ça le restera !

8.6. cinquième forme normale

On a vu que toute association ternaire ne se décompose pas forcément en deux projections. La question est que se passe-t'il si on utilise les trois projections?

Comme précédemment deux cas peuvent se présenter

.6.1. le cas où ça marche:

Modèle	moteur	couleur
306	bleue	diesel
306	verte	diesel
306	bleue	essence
306	verte	essence
605	rouge	essence
806	verte	diesel
806	bleue	diesel
806	blanche	diesel
806	verte	gpl

modèle	couleur
306	verte
306	bleue
605	rouge
806	bleue
806	blanche
806	verte

modèle	moteur
306	diesel
306	essence
605	essence
806	diesel
806	gpl

moteur	couleur
gpl	verte
diesel	verte
diesel	bleue
essence	verte
essence	bleue
essence	rouge
diesel	blanche

modèle	moteur	couleur
306	diesel	bleue
306	diesel	verte
306	essence	bleue
306	essence	verte
605	essence	rouge
806	diesel	verte
806	diesel	bleue
806	diesel	blanche
806	gpl	verte

8.6.2. le cas où ça ne marche pas:

modèle	moteur	couleur
306	diesel	verte
306	diesel	bleue
306	essence	verte
306	essence	bleue
605	essence	rouge
806	diesel	blanche
806	diesel	bleue
806	gpl	verte

Nota: plus de 806 diesel verte

modèle	couleur
306	verte
306	bleue
605	rouge
806	blanche
806	bleue
806	verte

moteur	couleur
gpl	verte
diesel	verte
diesel	bleue
essence	verte
essence	bleue
essence	rouge
diesel	blanche

modèle	moteur
306	diesel
306	essence
605	essence
806	diesel
806	gpl

modèle	couleur	moteur
306	bleue	diesel
306	verte	diesel
306	bleue	essence
306	verte	essence
605	rouge	essence
806	verte	diesel
806	bleue	diesel
806	blanche	diesel
806	verte	gpl

ERREUR

Comme au paragraphe précédent, on voit l'apparition **d'information erronées**

Définition 1

Une dépendance sera dite "de jointure" si **toutes** ses projections redonnent par composition l'association initiale.

Remarque:
Comme précédemment, l'association peut être de jointure, mais non décomposable si elle comporte une DF

Définition 2

5FN= 4FN + pas de dépendance de jointure ou il existe une définition fonctionnelle

Remarque comme pour 4FN, il faut distinguer les "dépendance de jointure occasionnelle" et "dépendances de jointure prouvées".

Seules celles de deuxièmes catégories sont **effectivement décomposables.**

CHAPITRE 7:
COMPLEXITE DES ALGORITHMES

Les données pouvant être très volumineuses, les temps de traitement peuvent devenir importants, voire prohibitifs, si de mauvais algorithmes sont appliqués à des données mal structurées. Deux types de traitement sont très fréquemment rencontrés: la recherc d'information dans une table et le tri des données

7.1. la recherche d'informations:

Divers types de problèmes se posent suivant le nombre de table, leur organisation, le type de données

7.1.1. recherche dans une table unique.

Enoncé du problème

Etant donné une entité A(a1,a2,. . . ak), on cherche toutes les occurrences telles que a1=x.

Dans un premier temps, la nature de x (numéro, chaine de caractère, etc…) est sans importance. Ce qui déterminera les performances sera l'organisation des données en particulier si elles sont triées ou non.

7.1.1.1. données non triées

Si les données ne sont pas triées par a1 croissants (ou décroissant), ou si elles le sont pour un autre attribut ai quelconque, une seule solution consiste à examiner successivement toutes les occurrences A[i] de A , dans l'ordre où elles se présentent dans la table A; on appelle ceci "balayer" la table. Il y a deux algorithmes selon qu'il faut chercher une seule occurrence (la première) ou toutes:

Une seule occurrence:

```
i=0;
répéter  i=i+1  jusqu'à  A[i].a1= x;
utiliser A[i].*
```

dont le coût est en moyenne de n/2 comparaisons

toutes les occurrences:

```
pour i=0 jusqu'à i=n faire
si A[i].a1= x alors utiliser A[i].*
```

dont le coût est de n comparaisons.

Ces algorithmes qui varient linéairement avec n deviennent vite très lourds

Remarque: le coût est ici exprimé en nombre de comparaisons entre attributs; mais le temps de calcul d'une comparaison peut être très variable suivant le type des données.

Exemple 1: comparaison de nombres entiers: très rapide, c'est une opération de base de l'ordinateur. (coût 1)

Exemple 2: comparaison de chaînes de caractères: le temps dépend de la longueur de la chaîne, et peut devenir important

Exemple 3: attributs non atomique (par exemple une adresse, une date), les temps deviennent très important.
Exemple 4 extrême! attributs de type image, par exemple rechercher une empreinte digitale ou la photo d'un criminel!

7.1.1.2. données triées

Dans le cas où on cherche une seule occurrence, l'algorithme le plus performant est celui dit par "dichotomie": il consiste à partager le tableau en **deux** parties égales , à chercher dans quelle partie se trouve l'information cherchée, et à recommencer avec cette partie:

a) sous forme itérative:

```
p=0;q=n;
répéter
        c=(p+q)/2;
        si A[c].a1 >x  alors p=c    {informationt dans la première moitié du tableau}
                       sinon q=c    {information dans la deuxième moitié du tableau}
jusqu'à  A[c].a1= x;
utiliser A[c]
```

b) l'algorithme peut se mettre sous forme récursive plus élégante (voir cours de programmation)

On cherche ainsi successivement dans des sous tableaux de dimension n, n/2, n/4, etc.. Au bout de $LOG_2(n)+1$ coupures successives, le sous tableau n'a plus qu'un élément, celui cherché.

Exemple: chercher une définition dans le petit Larousse (100 000 enregistrements)
Le dictionnaire étant trié par ordre alphabétique, le coût sera de **17** (car $LOG_2(100\ 000)<17$) et non 50 000 avec la première méthode. On voit qu'on a tout intérêt à avoir des données triées

Nota: S'il y a plusieurs occurrences à trouver, elles sont contiguës dans A; quand on en a trouvé une, il suffit de regarder "autour" pour trouver les autres.

1.1.1.3: technique de hachage

On suppose que les données Xi sont réparties en zones suivant une fonction caractéristique$Y=f(X_i)$ dite "fonction de hachage".

Exemple: le dictionnaire est réparti en zones caractérisées par la première lettre des mots: f(mot)="première lettre". Le livre comportera des "onglets" permettant de sélectionner rapidement une zone.

Plus généralement:
Etant donné un champ A de type quelconque, on définit une fonction de hachage y=f(A) dont les caractéristiques seront définies ci dessous (voir tri par hachage). En général f sera une fonction a résultat entier compris entre 1 et k, répartissant les occurrences en **k** zones de dimension moyenne **n/k.**

Etant donnée **x** l'information clé, la recherche se fera par:

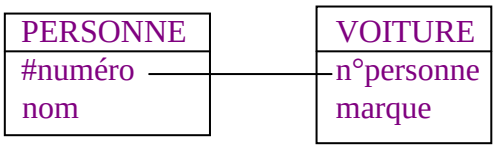
a) déterminer **y=f(x)**
b) chercher dans la zone n° y; cette recherche pouvant se faire par balayage ou dichotomie.

Exemple 1: les données dans les zones ne sont pas triées:
Coût de (a): négligeable
Coût de (b): n/k (et non n)

Exemple 2:les données des zones sont triées: (cas par exemple du dictionnaire)
Coût de (a): négligeable
Coût de (b): $LOG_2(n/k)+1 = LOG_2(n)- LOG_2(k)+1$
On remarque qu'on n'a pas gagné grand chose si **k est petit**.. par exemple pour la recherche dans le dictionnaire, le hachage fait gagner $LOG_2(26) =5$ opérations.

7.1.2. recherche dans plusieurs tables, cas des jonctions

exemple:



SELECT * FROM personne INNER JOIN voiture ON numéro=N°personne;

Soient A et B les deux tables de dimension respectivement n et p, la jointure se faisant sur les attributs a1=b1; la méthode est la suivante:

pour chaque A[i] de A on cherche dans B s'il existe des éléments j tel que B[j].b1=A[i].a1: c'est en fait un problème de recherche dont le coût est pour chaque ai:

- si la table B est triée coût= **$\log_2 p$**
- si la table B n'est pas triée: coût **p**

donc un coût total **n.p** ou **$n.\log_2 p$** on a donc intérêt à trier, c'est ce que fait systématiquement ACCESS

➔**coût total**= coût du tri + coût de recherche:

= **$p.\log_2 p + n.\log_2 p$** (et non n.p !)

nota: on voit que A JN B n'a pas le même coût que B JN A

Remarque:

supposons que A et B n'aient pas un identifiant unique:
soit la théta jointure

Select * from A inner join B on (a1=b1 OR a2=b2)

Pour chaque i on cherche

-quel sont les j tel que A[i].a1=B[j].b1 Ca va vite car B peut être trié par b1

-quel est le k tel que A[i].a2=B[j].b2 Ca va **lentement** parce que B ne peut être trié à la fois pour b1 et pour b2

Ce n'est pas comme ça qu'il faut s'y prendre, **solution:**

```
(Select * from A inner join B on a1=b1)
UNION
(Select * from A inner join B on a2=b2)
```

ici ACCESS fera un tri différent pour chaque sélect

7.2. Tri des données

On a vu que la bonne solution pour la recherche d'informations est de disposer de données triées. Quel en est le coût?

Exemple du tri de personnes par ordre alphabétique des noms.

7.2.1. Une méthode simpliste peut être imaginée:

Soit A1 le tableau non trié, A2 le tableau trié à construire.

a) chercher le première personne par ordre alphabétique, (comme le tableau est en désordre, cela peut prendre N opérations), l'extraire de A1, le ranger dans A2

b) recommencer avec donc un tableau A1 ayant une personne de moins.

Le coût sera N pour la première fois, N-1 la deuxième , N-2 la troisième etc.
Coût total n+(n-1)+(n-2)+..... 2+1= n.(n+1)/2 . Ainsi trier un tableau d'un million de personnes demanderait 500 milliards d'opérations.

7.2.2. Une méthode plus rusée sera de copier ce qu'on ferait à la main sur des fiches papier:
on fait deux tas: à gauche les noms commençant par a, b, c,...k, l, et à droite les noms commençant par m, n, o,...y, z Cette opération demande d'examiner chaque fiche, cela coûte N on obtient ainsi deux tas d'approximativement N/2 fiches.
on recommence le tri de chaque tas en coupant chacun en deux; cela coûte N/2 pour chaque tas (donc encore N pour le total). Et ainsi de suite jusqu'à avoir des tas d'une seule fiche.

Nota:

a) à chaque étape, on a un même coût de N

b) On remarque qu'on ne peut couper en deux que Log_2 N fois,

le coût totale est donc N. Log_2 N.

Exemple: trier un million d'informations: le coût total sera de 20 millions d'opérations (et non 500 milliards)!

Le programme sera le suivant:

a) supposons l'existence d'une fonction F(X,Y) calculant à partir de deux noms X et Y un nom intermédiaire entre X et Y de telle sorte qu'il y ait à peu près autant de noms entre X et Z qu'entre Z et Y. (exemple entre "albert" et "zoe" on trouverait "lucie")

b) l'algorithme de tri sera:

X= "aaaa"; Y= "zzzz";
a=1; b=N;

calculer une information "médiane"F(X,Y)
examiner un à un tout les éléments du tableau et les classer en deux zones de taille approximativement N/2 en comparant à la valeur médiane.
entre a et p, les éléments inférieurs à F(X,Y)
entre p et b, les éléments supérieurs à F(X,Y)
{coût N comparaisons et rangements}

recommencer récursivement pour les deux moitiés avec respectivement a=1 b=p pour la première,a=p+1, b=n pour la deuxième, jusqu'à a=b

Nota:

Un tel tableau trié peut être soumis à mises à jour (ajouts, suppressions). Il serait alors idiot de faire des opérations du type **{ajouts en fin de tableau+ tri}.** On préférera en général faire deux tableaux:

un tableau A1 principal, contenant toutes les informations **anciennes**,

un tableau A2 auxiliaire plus petit contenant les informations **nouvelles**, trié **fréquemment** (mais rapidement)

Quand ce dernier tableau devient important, on fusionne les deux par "interclassement"

7.2.3. Interclassement:

Etant donnés deux tableaux triés, A et B de dimensions p et q, on veut les fusionner en un seul tableau C trié.

Méthode:

Supposons que le premier élément de A soit inférieur au premier élément de B

1) Tant que ([i]>B[j]) et (B non vide) faire {mettre A[i]dans C, i=i+1}
2) Tant que (B[i]>A[j]) et (A non vide) faire {mettre B[j]dans C, j=j+1}
Recommencer en 1 jusqu'à épuisement des deux tableaux

Coût de l'opération: p+q comparaisons.

7.2.3. Hachage (hasch coding en anglais)

Problème: le tri, même par dichotomie, peut être trop lent)pour être envisagé dans sa globalité. On lui préférera souvent une technique moins coûteuse: celle de hachage décrite au paragraphe 7.1.3.

ANNEXE 1

METHODE NIAM
(Nijssen Information Analysis Method).

1 Le niveau conceptuel:

Au départ: la langue naturelle qui exprime un problème sous forme de phrases indépendantes.

Exemple:

"chaque étudiant est inscrit à un département qui a un responsable"

Cette phrase n'étant pas atomique peut être décomposée:

"chaque étudiant est inscrit à un département"
"chaque département a un responsable"

Cette fois les phrases sont atomiques: on ne peut les décomposer sans changer leur sens.

Nota: Eventuellement on complète ces phrases atomique par des précisions du genre:

"chaque étudiant repéré par son numéro est inscrit à un département repéré par son nom" .

Une telle phrase n'est plus atomique et peut se décomposer en :

"chaque étudiant est inscrit à un département"
"un étudiant est repéré par son numéro"
"un département est repéré par son nom"

Apparaissent alors des notions diverses:

Des "concepts" ou objets **"non-lexicaux"** (NOLOT= Non Lexical Object TYPE):
l'étudiant, le département...

Des objets **"lexicaux "** (LOT= Lexical Object Type) décrivant les concepts:
le numéro, le nom, etc.. On les qualifie de "lexicaux" parce qu'on peut en faire un catalogue (lexique)

Des **relations** entre LOT et NOLOT:
un étudiant à un numéro, un numéro est affecté à un étudiant,

Des **relations** entre NOLOT

un étudiant appartient à un groupe.

1. Le modèle:

1.1. Les non lexicaux représentés par des cercles en traits pleins: exemples.

1.2. les lexicaux

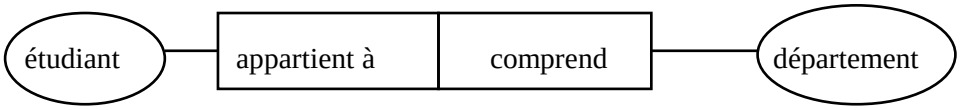
Entiers, chaînes de caractères, images, etc...
seuls ou à plusieurs, ils définissent les NOLOTs (comme les attributs définissent les entités en Merise)
ce sont des "terminaux": il ne sont pas définis par autre chose

représentation: cercle en pointillé:

1.3: Les idées:

ce sont des relations binaires entre NOLOTs, représentées par des rectangles

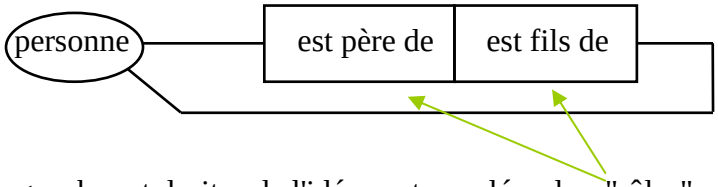
exemple:



ce qui se lit: *"un étudiant appartient à un département"*
"un département comprend des étudiants"

Nota: une idée peut relier deux occurrences d'un même NOLOTs

exemple

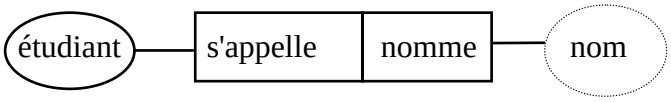


définition: les parties gauches et droites de l'idée sont appelées des "rôles".

1.4: Les "ponts" (ou "pont de dénomination").

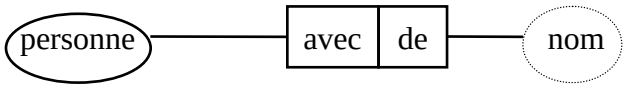
Ce sont des relations entre LOTs et NOLOTs

exemple



ce qui se lit: *"étudiants'appelle X" ou "X nomme l'étudiant"* (X étant une chaine de caractère ou occurrence de nom)

Sémantique: en général une idée est formée de deux rôles correspondant à des groupes verbaux (exemple "est père de", "est fils de") mais pour les ponts, on simplifie parfois en ne mettant que des préposition: exemple:



"une personne **avec** son nom" ou "un nom **de** personne"

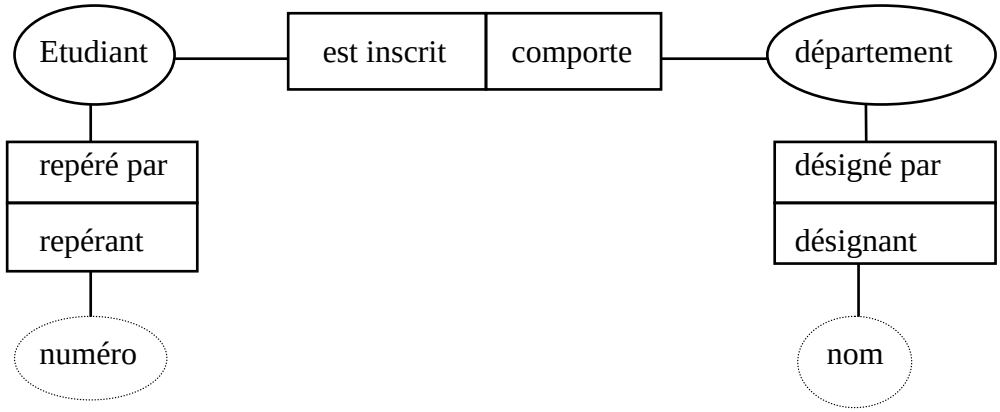
Nota:

a) il n'y a jamais de ponts entre LOTs : ce sont informations "terminales" qui se suffisent à elles mêmes. Cela suppose que l'information du LOT soit atomique (attention pour les dates par exemple)

b) pas de différence fondamentale entre idées et ponts, excepté la nature des extrémités.

c) la notion d'idée ou de pont n'est pas équivalente à celle de relation en Merise; en particulier l'idée relie deux NOLOTS et deux seulement.

1.5: Exemple de schéma conceptuel:



2. Les contraintes.

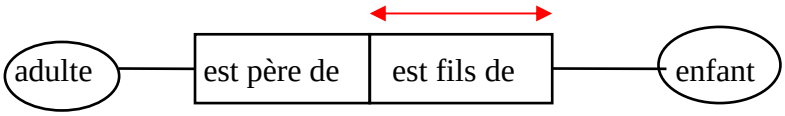
Le caractère binaire de l'idée est très restrictif: la notion est beaucoup moins puissante que la relation de Merise par exemple qui peut être ternaire ou plus. Or dans Merise, on constate que toutes les relations ne peuvent pas forcément être projetées pour donner des relations binaires sans introduire des informations anormales. Ce défaut de l'idée doit alors être compensé par de nombreuses informations supplémentaires: les contraintes.

Ce sont des informations sur la nature des idées ou des ponts, informations qui devront être vérifiées , par exemple à la saisie.

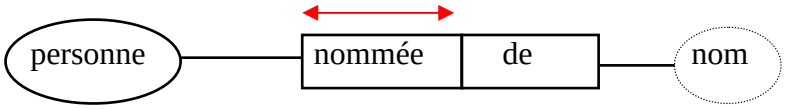
2.1. unicité du lien rôle-NOLOT : notée ⟷ au dessus du rôle.

La notion est équivalent à celle de cardinalité (x,1) de MERISE.

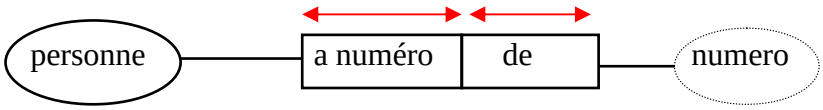
exemple 1: "un enfant a au plus un père".(mais un adulte peut avoir plusieurs enfants et un enfant peut être né de père inconnu)



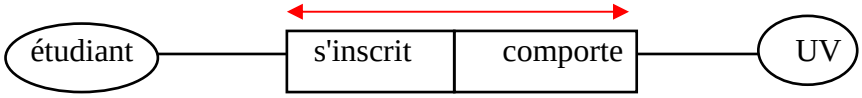
exemple 2: "une personne a un nom et un seul" (mais un nom peut correspondre à plusieurs personnes).



exemple 3: biunivocité: "une personne est caractérisée par son numéro INSEE".



exemple 4: aucune contrainte: "un étudiant s'inscrit à des UV"(en étudiant s'inscrit à plusieurs UV, une UV comporte plusieurs étudiants).



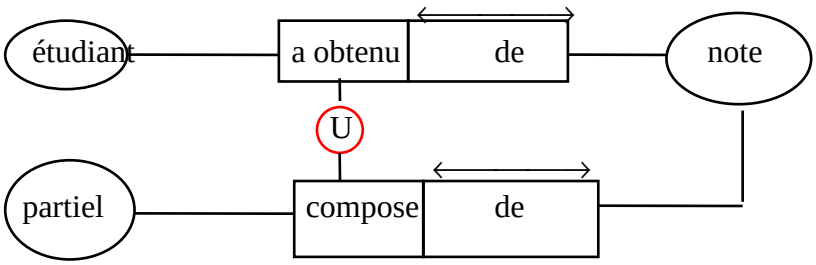
.

Nota: l'absence de tout ⟷ sur l'idée est une erreur et non l'absence de contraintes.

2.2. contraintes d'unicité entre rôles.

Ces contraintes notées U relient les rôles d'idées ou ponts différents:

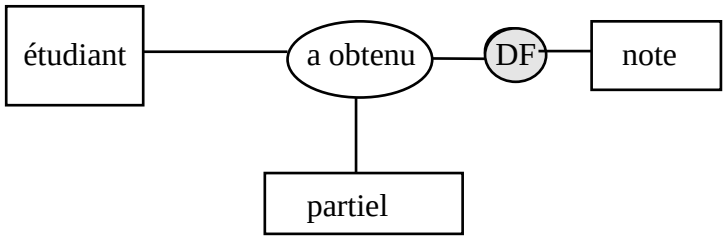
exemple base de donnée des notes d'étudiants.



la note est unique pour le concept "étudiant" et le concept "partiel". On peut dire aussi: "pour un partiel donné et un étudiant donné, il y a une seule note".

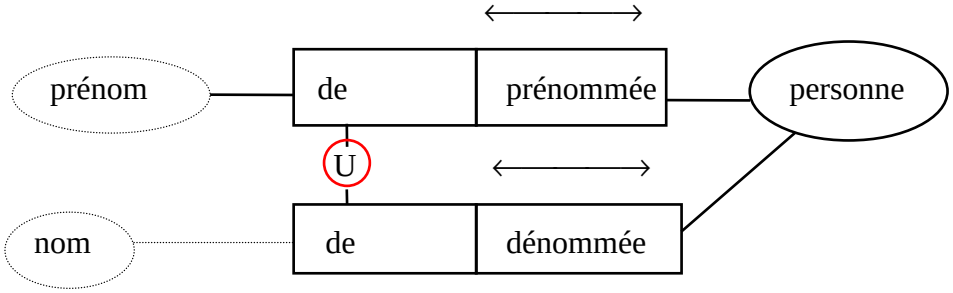
Par contre il faut noter qu'un partiel a plusieurs notes et un étudiant plusieurs notes

Nota: c'est beaucoup plus détaillé qu'en Merise où, grâce aux relations ternaires, on aurait:



De toute évidence, si on projetait cette relation en deux relations binaire, on perdrait de l'information! Il faut donc préciser quelque chose dans le schéma.

On aurait le même type de contraintes avec des ponts; exemple

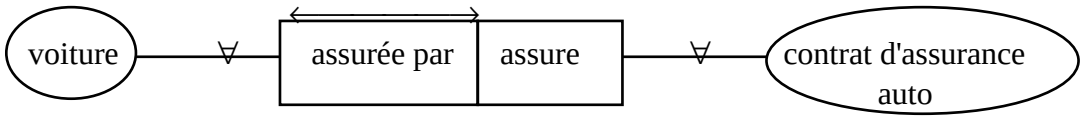


qui se lit: "pour un prénom donné et un nom donné on a une seule personne".

2.3. Contraintes de totalité entre NOLOTs et rôles.

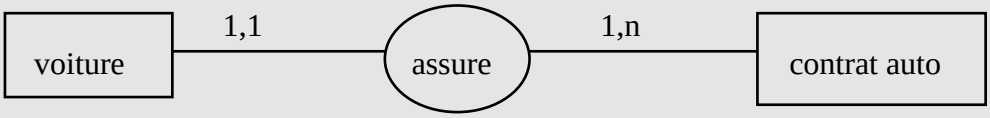
Elles traduisent l'obligation pour toute occurrence d'un NOLOT d'intervenir dans un rôle d'idée ou de pont ; cette contrainte est notée ∀. Cette contrainte est équivalente à la cardinalité (1,x) de MERISE

Exemple

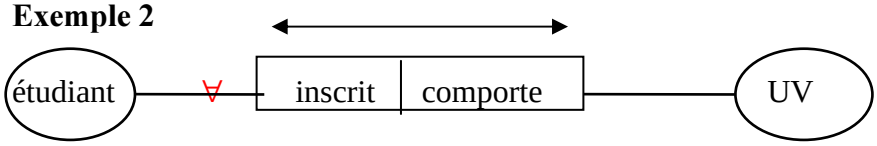


" toute voiture est assurée par un contrat exactement, tout contrat concerne une voiture (éventuellement plusieurs)".

Ceci serait l'équivalent de MERISE:

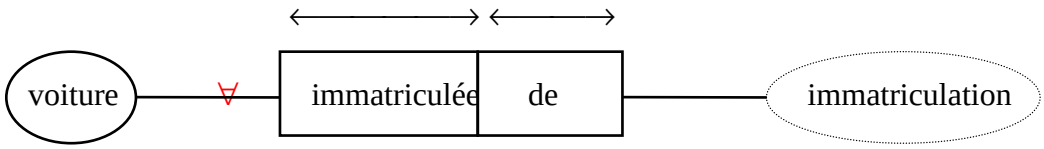


Exemple 2



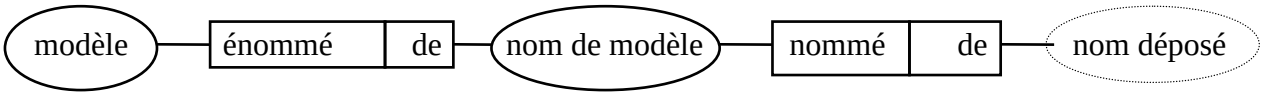
"tout étudiant est inscrit à au moins une UV, mais certaines UV ne comportent pas d'étudiants".

Remarques:
a) Cette contrainte ne s'applique pas aux LOT, car par définition, tout LOT est obligatoirement associé à un concept:



il n'y a évidemment pas d'immatriculation sans voiture: le signe∀ et inutile.

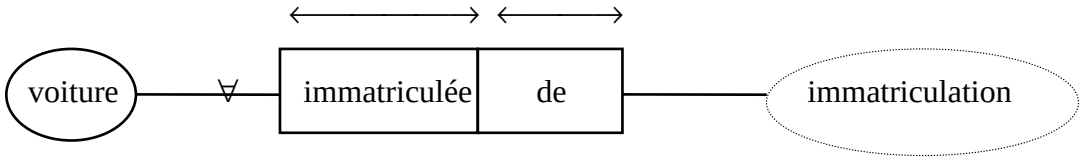
b) Parfois des LOT ne sont apparemment pas affectés; par exemple le cas de noms déposés d'avance (les noms de la forme X0Y des modèles Peugeot); il faut alors faire intervenir un NOLOT supplémentaire:



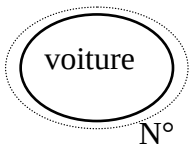
L'existence du nom déposé implique l'existence d'un nom de modèle, bien que le modèle puisse ne pas encore exister.

Fusion LOT/NOLOT:

Dans les cas de bijection LOT - NOLOT on fusionnera les deux :exemple



s'éxprimera de manière plus condensée par



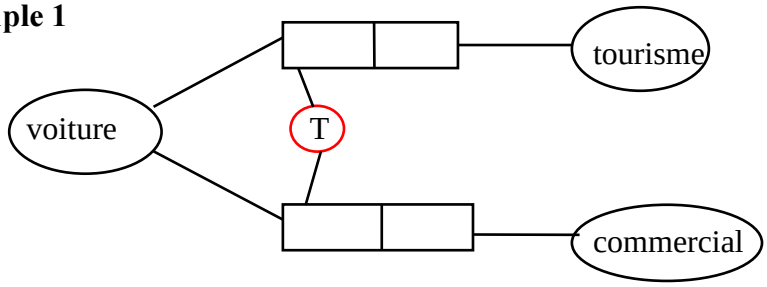
Le numéro de la voiture devient donc son identifiant

NOTA:
compte tenu des 4 possibilité de contraintes sur les liens, **il y a 16 types d'idées .**

2.4. contraintes de totalité entre rôles.

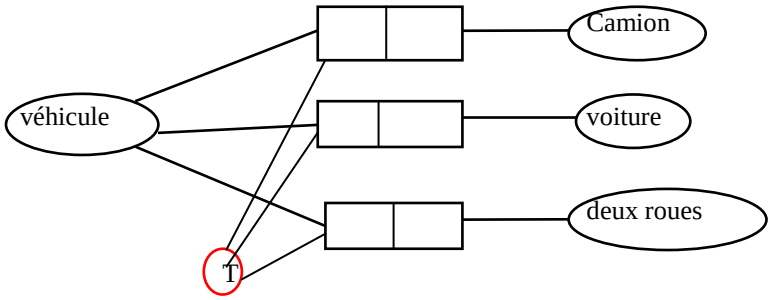
Elles expriment le fait qu'un NOLOT est soit d'un type soit de l'autre (éventuellement les deux)

exemple 1



exprime que toute voiture est soit à usage commercial soit à usage tourisme, éventuellement les deux.

La contrainte de totalité peut évidemment porter sur plus de deux NOLOTs

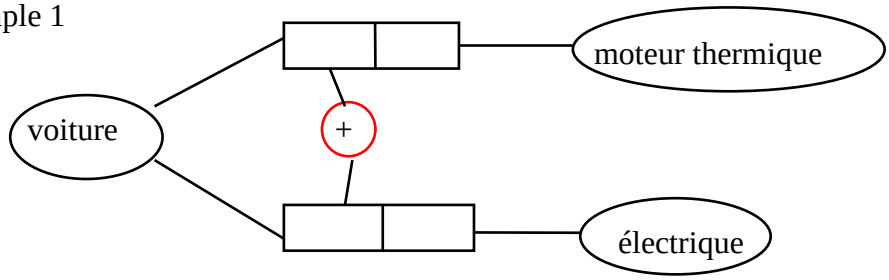


La réunion de l'ensemble des camions, de l'ensemble des voitures, de l'ensemble des deux roues forme l'ensemble des véhicules

2.5. contraintes d'exclusion entre rôles

Elles expriment le fait qu'un NOLOT est d'un type ou d'un autre, mais pas les deux (éventuellement d'aucun)

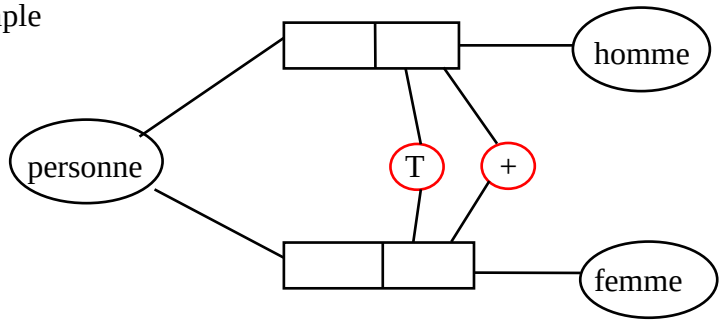
exemple 1



une voiture peut avoir un moteur thermique ou électrique (mais pas les deux; il peut y avoir des voitures sans moteur (à pédales)

NOTA les contraintes d'exclusions et de totalité peuvent coexister.

exemple

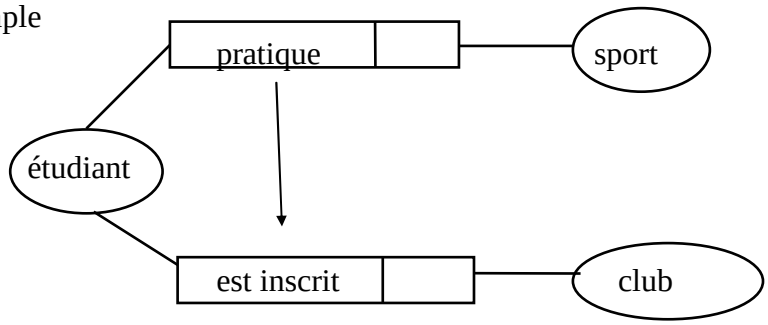


toute personne est soit un homme soit une femme, mais pas les deux.

2.6. Contraintes d'inclusions entre rôles.

Elles expriment qu'un rôle en implique un autre:

exemple

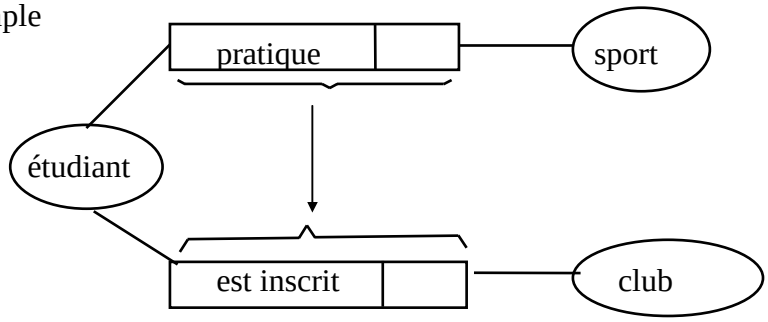


L'ensemble des inscrits à un club contient l'ensembles des étudiant pratiquant effectivement dans ce club; (pratique implique inscrit).

2.7. Contraintes d'inclusions entre idées

Elles expriment qu'une idée (les deux rôles) en implique une autre:

exemple



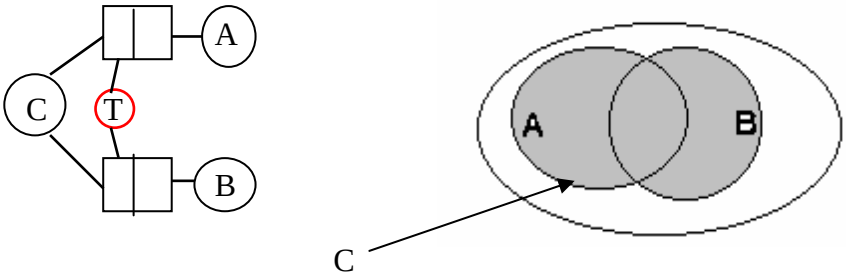
Ici, un étudiant pour pratiquer un sport doit être inscrit à un club. (mais il peut faire du ski et être inscrit au judo).

2.8. Expressions logiques des contraintes entre rôles

Rappel: les containtes s'expriment en terme de **relations ensemblistes.** Ces relations ne peuvent concerner que des ensembles de même nature

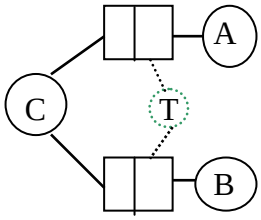
2.8.1. contrainte de totalité:

L'ensemble des C est formé de(l'ensemble des A) **union** (l'ensemble des B)
Ou encore:
Les C qui sont relié à A "plus" l'ensemble des C qui sont reliés à B forment la totalité des C



Remarque 1: la contrainte porte sur des rôles ayant une même "charnière" C. Expliquons nous:
Soient X et Y les idées reliant respectivement C à A et C à B.:
X est un ensemble de couples {Ci, Aj}
Y est un ensemble de couples {Ck,Bl}
La contrainte exprime que C est l'union de l'ensemble {Ci} et de l'ensemble {Ck}. Il s'agit de l'union d'ensemble de même nature.

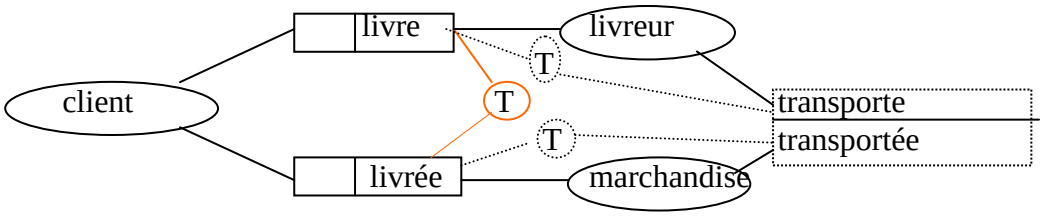
Remarque 2: on pourrait imaginer des contraintes portant sur les rôles de droite:



En principe, une telle contrainte n'a pas de sens: elle impliquerait l'union de deux ensembles {Aj} et {Bl}**de nature différente,** ce qui n'a pas de sens!

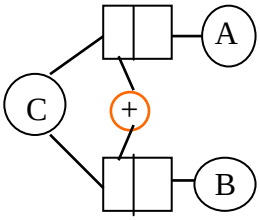
Par contre, on pourrait imaginer une contrainte de totalité entre l'existence d'un A relié à C et l'existence d'un B relié à C exemple:

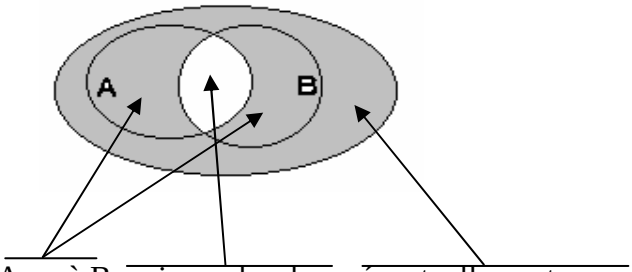
Dans l'exemple suivant, imposer une contrainte de totalité:



Mais ceci implique probablement une idée sous-jacente: "toute marchandise est transportée par le livreur". On pourrait alors mettre les contraintes en pointillé qui portent sur des ensembles de même nature.

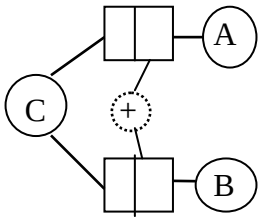
2.8.2. contrainte d'exclusion





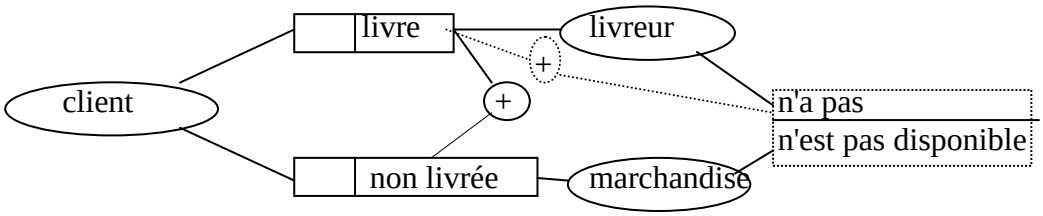
C= $\overline{A \cap B}$ = C est relié à A ou à B mais pas les deux, éventuellement aucun.

Attention: Comme dans le cas précédent, une telle contrainte portant sur les rôles de droite n'a pas de sens:



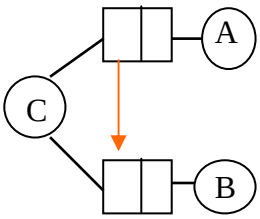
On ne peut pas faire l'intersection de deux ensembles de nature différente!

Exemple:



On cherche à dire que soit le client est livré, soit la non disponibilité de la marchandise lui est signifiée, mais pas les deux. Le schéma est probablement incomplet, la partie en pointillé doit être rajoutée.

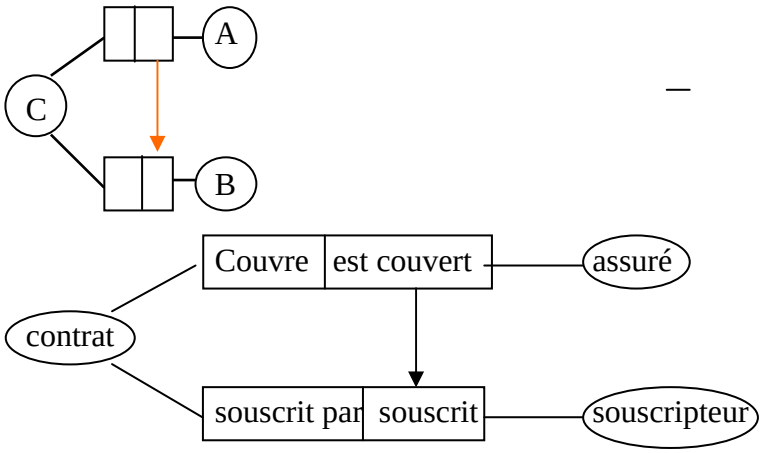
2.8.4. implication:



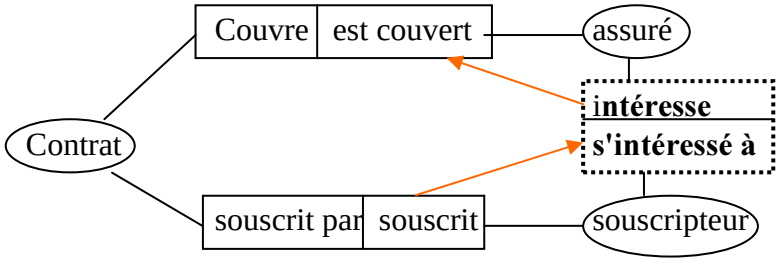
(C est non relié à A)ou (C est relié à B): C= $\overline{A} \cup B$

Nota: cette fois encore une implication des rôles droite **pourrait** être envisagée

Exemple

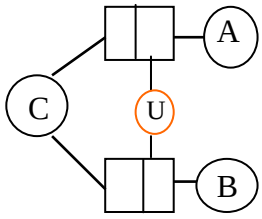


On ne s'intéresse pas au fait qu'un contrat couvrant un assuré implique que le contrat ait un souscripteur, mais au fait que l'**existence d'un assuré implique un souscripteur.** L'idée n'est pas idiote, mais seulement incomplète. Il y a quelque part une idée sous-jacente: **le souscripteur s'intéresse à l'assuré:**



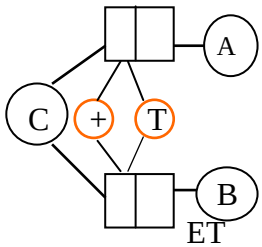
Maintenant chaque implication concerne des ensembles de même nature

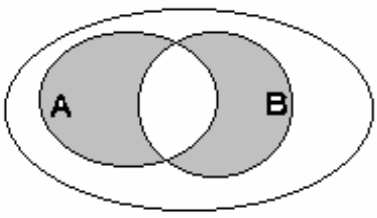
2.8.5. Unicité



pour un A et un B on a un seul C: $C=A \cap B$?
Nota: contrainte toujours associée au rôle de droite (implicite à gauche)

2.8.3. combinaison de deux contraintes: une contrainte ET l'autre:

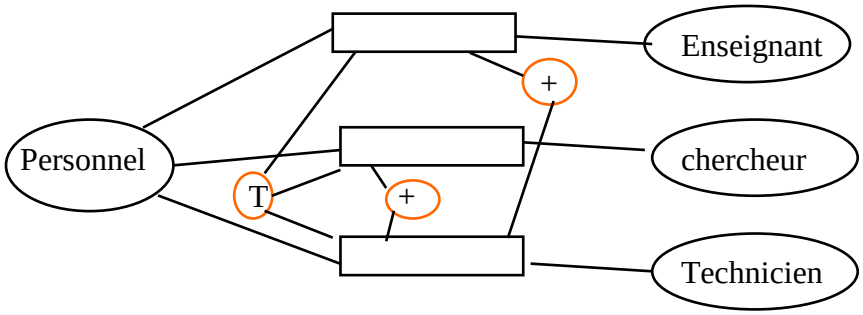




$C = (A \cup B) \cap \overline{(A \cap B)} == (A \cup B) \cap (\overline{A} \cup \overline{B}) = \overline{A} \cap B \cup A \cap B$ c'est le ou exclusif

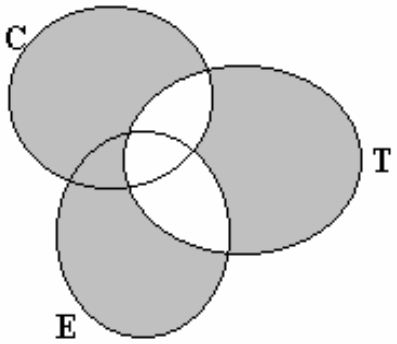
2.8.4. On peut ainsi envisager des contraintes complexes s'exprimant par des fonctions booléennes:

exemple: un personnel de l'UTBM peut être soit (un enseignant ou un chercheur) soit un technicien(il peut être enseignant et chercheur, mais pas enseignant et technicien ni chercheur et technicien).



$P=(E \cup C \cup T) \cap \overline{(E \cap T)} \cap \overline{(C \cap T)}$

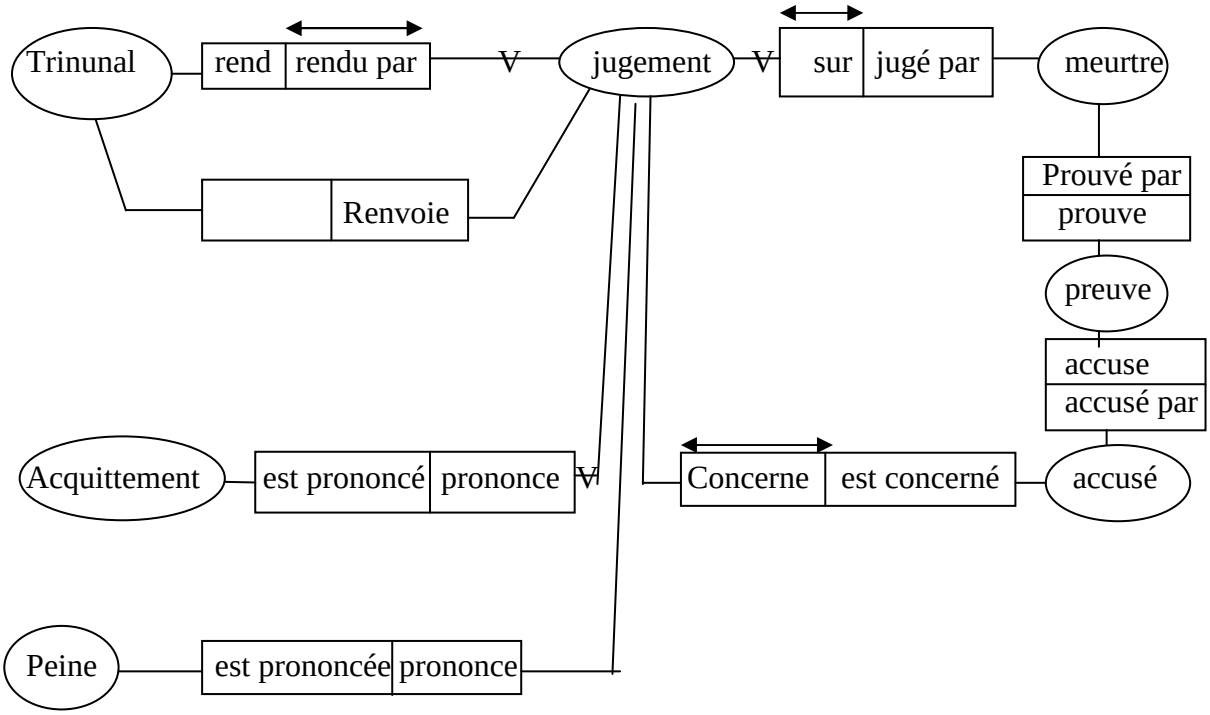
$= C \cap \overline{T} \cup E \cap \overline{T} \cup T \cap \overline{E} \cap \overline{C}$



2.8.5. Décomposition d'une contrainte globale

Le problème s'exprime en général sous la forme inverse, on sait globalement ce qu'on veut, il faut trouver les contraintes élémentaires.

Exemple pour une personne donnée, un crime donné, le tribunal se déclare incompétent et renvoie la décision à un autre tribunal, ou prononce l'acquittement en l'absence de preuve ou une peine mais pas les deux :



La fonction s'exprime en générale sous forme d'une "somme de produits":

Jugement= renvoi ou (preuve et peine) ou (non preuve et acquittement)

Simplifions les notations et passons pour la simplicité en algèbre de Boole:

Jugement= A ou (B et C) ou (non B et D)

= A + B C+$\overline{B}$ D

Or on sait que les opérateurs + et . sont doublement distributifs l'un par rapport à l'autre:
En particulier X+ X Z= (X+Y)(X+Z) pour tout X,Y,Z

Donc jugement= (A+B)(A+C) +$\overline{B}$ D= (A+B+$\overline{B}$)(A+B+D)(A+C+$\overline{B}$)(A+C+D)
On peut simplifier en remarquant que A+B+$\overline{B}$ = A+1 = 1 et peut donc être éliminé

Jugement=(A+B+D)(A+C+$\overline{B}$)(A+C+D)

= (renvoi ou preuve ou acquitement)(renvoi ou peine ou non preuve)
(renvoi ou peine ou acquitement)

donc trois contraintes: totalité entre {renvoi, preuve, acquitement}
totalité également entre {renvoi, peine,acquitement)
implication non_preuve➔(renvoi ou peine)

THEOREME

Toute fonction exprimée en somme de produits peut se mettre en produit de sommes, donc toute contrainte complexe peut se décomposer en contraintes élémentaires.

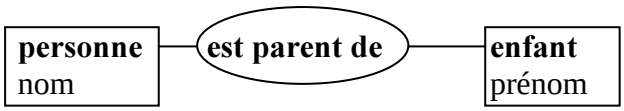
Nota: par contre le théorème ne dit pas que les contraintes élémentaires seront toujours parmi celles normalisées.

2°partie
TD BD40

2.1.notion d'association

comparer l'encombrement de deux représentation de l'état civil d'une personne

a) version EXCEL:
personne={nom, enfant1, enfant2, …enfant n}
et
b)version ACCESS:



-discuter sur le nombre d'enfants possibles dans les deux cas
-faire ressortir l'intérêt de la clef numérique:
-si N est le nombre de personne, P caractères par nom, Q le nombre max d'enfants, R relations de parentés, calculer l'encombrement dans les deux cas

-prendre un exemple: N=1000, P=20, Q=10, R<<1000.

2.2. modèle Merise: gestion des commandes:
Les clients d'une entreprise émettent des commandes comportant des produits

trouver le schéma,
discuter les cardinalités des liens.
Que faire si le client change d'adresse.
Expliquer comment en ACCESS on réalisera les associations (MLD).

2.3. Base de donnée "informatique du GI"

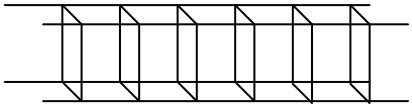
Il s'agit de gérer toutes les informations relatives au matériels et logiciels du département, pour répondre à des questions du genre:
"combien avez vus de PC ou de SUN, pour quelle somme, quel est leur vétusté, ..."; "quels sont les périphériques" , "quels sont les logiciels temps réel", "où sont ils", "où peut-on installer tel logiciel", etc...

définir les cardinalités des liens.
bien faire la distinction entre deux approches:

a) logiciels et matériels sont des objets élémentaires (un logiciel avec son n° de licence, un poste particulier
b) logiciels et matériels sont des classes : exemple les compilateurs C, les PC de la salle X, etc..

2.4. TD "Suivi de fabrication"

Une entreprise fabrique des "armatures à béton": 4 barres reliées par des cadres:



Les barres et les cadres sont découpées dans des barres de fer en bobines vendues par un fournisseur, les cadres sont pliés puis soudés aux barres, les armatures sont livrées à des clients. Le problème est de pouvoir répondre aux réclamations des clients: qui a fabriqué quoi, avec quoi?

Approche 1: une entité par type d'objet, une association par opération.

définir les cardinalités, en se posant les questions: "est ce que chaque objet est identifié ou est-ce qu'on travaille par "lots", est-ce que toutes les opérations sont faites? (réponse: non, il y a des stocks intermédiaires➔ attention aux cardinalités)

approche 2: fusionner les entités en ajoutant un "type d'objet" ou type de machine
fusionner les associations en introduisant un "type d'usinage"

2.5. BD40 Sujet de TD N°3

Base de données textuelles: Il s'agit de mémoriser les informations données par un texte en langue naturelle, pour pouvoir répondre à des requêtes relatives à ce document.

Exemple:
phrase : *"le logiciel est installé par l'ingénieur sur le disque dur"*
question: "*où est le logiciel*?", *qui installe le logiciel*?", etc…
Il faut donc mémoriser les mots de la phrase avec leur rôle (sujet, verbe, complément, etc…) et leurs relations entre eux. Attention, il ne s'agit pas de mémoriser le style: une phrase telle *que "sur le disque dur est installé le logiciel par l'ingénieur*" contient la même information que la phrase précédente.

Question 1: faire le modèle sujet/verbe/complément.
Discuter le besoin d'avoir un identifiant pour les noms

Est-il intéressant de faire ressortir des entités telles que "sujet", "complément", "groupe verbal", etc..

Remarque: le rôle des mots principaux (verbes noms) sont définis par des prépositions:
Exemple *"sur le disque dur* ➔ *disque* est complément de lieu

Question 2: introduire les autres types de compléments:
Nom complément de nom: *traitement de texte*
Verbe complément de nom: *machine à écrire,*
Verbe complément d'adjectif*: prêt à travailler*

Question 3 Ne pas oublier
a) les adverbes:
type 1 lié à adjectif: "très rapide"
type 2: lié au verbe: "exécuter rapidement" et surtout "ne marche pas"

b) les "déterminants": le, la, les, ..mon, ma, …cet, cette, certains, tous, ..

c) les conjonctions et, ou

Question 4: comment traiter les propositions subordonnées?

question 5
discuter de la représentation des informations. Exemple les verbes:
- représentation 1: tout le groupe verbal est présent dans l'entité
  exemple "l'ordinateur *a été acheté* à bas prix"
- représentation 2:

on stocke l'infinitif du verbe, puis des informations du type "voie passive", passé composé, 3° personne du singulier
Discuter l'intérêt des deux: en particulier la forme 2 permet des questions approximative du genre "qu'est ce qui est acheté à bas prix" ➔ réponse, "l'ordinateur", bien que les temps ne concordent pas

2.6. Méthode NIAM: Base de données "enquête policière".

On s'intéresse à des enquêtes policières concernant des délits commis par des truands. Il y a plusieurs formes de délits (crime, vol, chantage...); ils donnent lieu à des indices (armes du crime, empreintes, etc..), ils impliquent des personnes (auteurs, victimes, témoin, complices, receleur,...). Les truands ont une situation (libre, en taule, en cavale,...), des antécédents, des relations. Il peut y avoir eu procès et condamnation, donc un casier judiciaire, etc. On peut broder à l'infini.

1) Faire le modèle Merise:, et en particulier définir les cardinalités. Il y a des contraintes externes aux cardinalités: (exemple on ne doit pas être condamné pour un crime qu'on n'a pas commis) mais pas l'inverse: on peut être criminel et ne pas être jugé.

2) Faire le Modèle NIAM
Approche 1:
a) on décompose les associations n-aires en idées binaire.
b) Définir les cardinalités:
c) Il faut faire intervenir des contraintes; exemple:
"pour un crime donné et une personne donnée, il y a une seule peine"
la comdannation implique qu'il y a crime (mais pas l'inverse); elle implique l'existence de preuve
etc…

Méthode 2:

Etant donné que seuls les NOLOTS peuvent avoir plus de 2 liens, il faut transformer les association ternaires en NOLOT et non en idées

Trouver les contraintes.

BD40 TD NIAM/contraintes

Déterminations de contraintes élémentaires à partir de contraintes globales

Exercice 1

Pour recruter des personnes (P) on désire qu'elles aient deux compétences au moins parmi les trois: technique (T), administratif (A), commercial(C)
a) modèle NIAM
b) trouver la contrainte générale:

P= A∩C ∪ A∩T ∪ C∩T plus lisible sous forme algébre de Boole :

P= AC + CT + AT

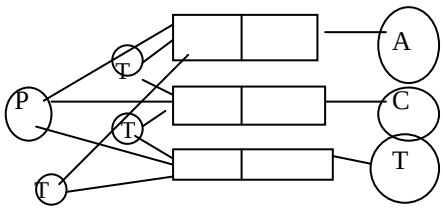
c) passer à une expression "produit de somme"

c1) rappel d'algèbre: double distributivité:
(a+b)c= ac+bc et symétriquement:
c2) ab+c=(a+c)(b+c)

c3) faire ressortir la symétrie entre et/ou
c4) développer :

P=(a+c)(c+t)(a+t) = (c+at) (a+t) d'après c2 = ac+ct+at d'après c1
Donc on sait faire de même en permutant et et ou
P=ac+ct+at=c(a+t)+at d'après c1 = (a+c)(c+t)(a+t) d'après c2

Donc la contrainte est le et de trois contraintes de totalité entre rôles



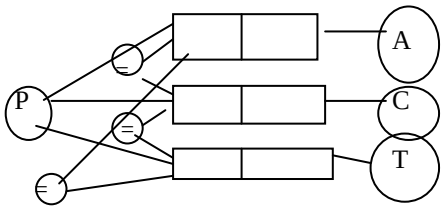
Exercice 2
Même question avec contrainte globale: P les trois spécialités ou aucune

P= A C T + A $\overline{C}$ $\overline{T}$ $\overline{}$ $\overline{}$ qui développé comme précédemment donne

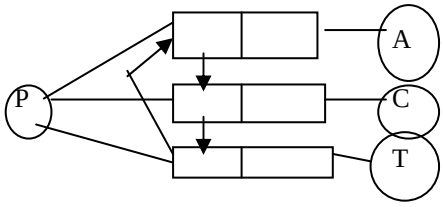
= (A+C)$\overline{(A+C)}$(A+$\overline{T}$)(A+T)(C+$\overline{T}$)$\overline{(C+T)}$ donc $\overline{6}$ contraintes d'implications
(A ⟶C)(C ~~A)~~ ▶....

Deux facons de voir les choses:

Ou on constate que
(A ⟶C)(C ~~A)~~ ▶ (A=C) et de même pour les autres couples de contraintes donc on a 3 contraintes d'égalité

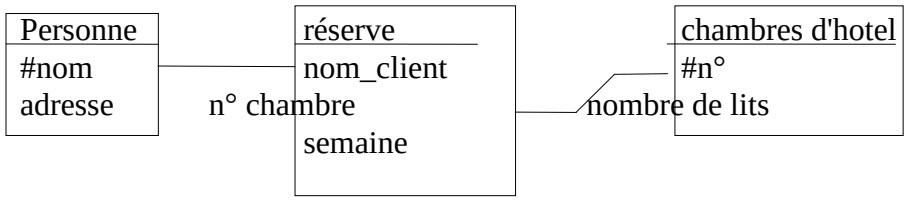


ou bien on constate
que la contrainte globale s'exprime aussi P=(A➔C)(C➔T)(T➔A) (chacun, directement ou indirectement implique les deux autres)



BD40 TD sur SQL,

Base de données des réservations de chambres d'hotel: en ACCESS



Problème 1: faire une requête sélection qui affiche les chambres disponibles une semaine donnée (infaisable en mode graphique)

1.1. faire une requête R1 qui affiche les chambres occupées. Observer l'instruction SQL. Faire ressortir le fait que "SELECT " définit un ensemble E, et que cet ensemble peut donc intervenir dans une autre instruction.

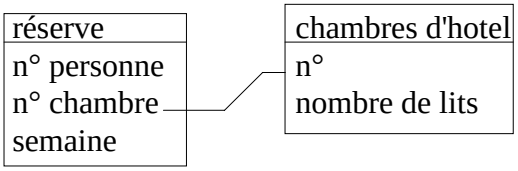
1.2. Faire la requête R2 qui selectionnera les chambres d'hotel "NOT IN E".

a) faire une requête qui sélectionne toutes les chambres de l'hotel

b) enlever les chambres réservées cette semaine:

1.2. constater qu'il y a deux versions:

a) en laissant la jointure réserve.n°chambre —— chambre.n°



qui génère une clause INNER JOIN.

b) Trouver l'erreur, la corriger

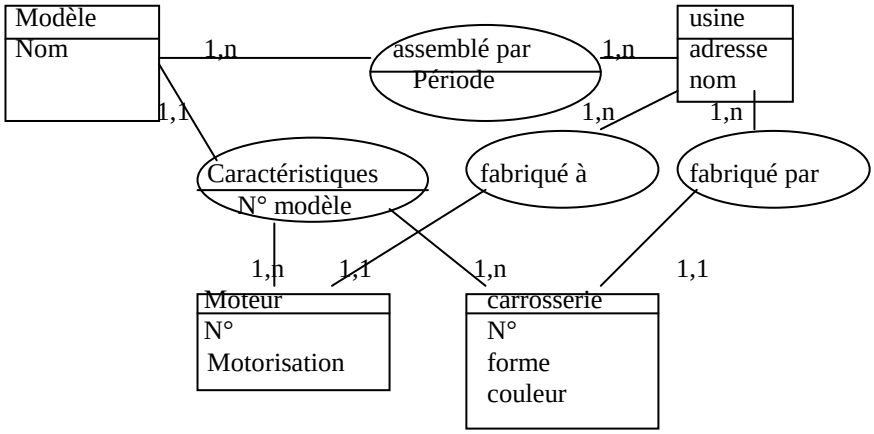
c) Discuter la solution:

```
SELECT DISTINCTROW [chambres d'hotel].n°
FROM [chambres d'hotel], réserve
WHERE ((([chambres d'hotel].n°) NOT IN
(SELECT réserve.[n° chambre] FROM réserve
WHERE ([semaine]=[quelle semaine?])));
```

Problème 2: annulation des réservations: il ne faut pas enlever les réservations de la table car on veut garder un historique. faire une saisie contrôlée: créer une table des annulations de réservations: il y a une contrainte: les annulations doivent correspondre à des réservations (même chambre, même semaine, même personne)

2.6. Algèbre relationnelle

Base de données des modèles de voiture d'un constructeur auto. Une voiture se caractérise par son nom (ex: Safrane) son type de carrosserie (ex: break), sa motorisation (ex: turbo diesel, 16 soupapes), l'année du modèle. On dispose du MCD suivant:



a) faire une requête SQL R1 qui trouve où sont assemblés les modèles, puis la requête R2 qui trouve où sont fabriquées les carrosseries et les moteurs.
b) Faire une requête R utilisant R1 et R2 qui trouve les modèles ne nécessitant pas un transport de la carrosserie
c) Ecrire cette requête en algèbre relationnelle .
d) La simplifier, (donner éventuellement les graphes descriptifs)
e) Ecrire la requête simplifiée en SQL

TD sur la minimisation des Bases de données

Enoncé du problème:

Quand on définit une BD, le choix des DF est difficile ; par exemple, on peut avoir ou non des DF directes, des DF élémentaires, etc… Il n'est pas évident du tout que le choix initial soit optimal

Modélisation des DF
Chaque DF s'exprime en terme d'implication de connaissance : à tout attribut A source ou résultat de la DF on associe une variable booléenne a: a=1 si A est connu, sinon a=0
Une DF {A,B,C..}➔ X s'exprime donc sous la forme booléenne: e= a.b.c…➔x ou encore:

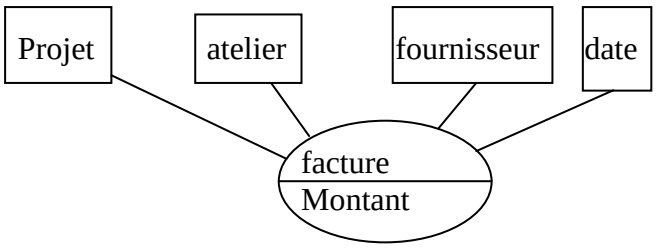
$$e=\overline{a.b.c.}\ldots+x = \bar{a} + \bar{b} + \bar{c} + \ldots + x$$

Fonction caractéristique des DF:

C'est l'assertion "toutes les DF ", donc le produit E des e_i.

Exemple si on a les DF suivantes (A,B)➔C et (C,D)➔D, on a $E=(\bar{a}+\bar{b}+c)(\bar{c}+\bar{d}+e)$

Cette fonction booléenne a plusieurs formes possibles; une forme minimale (en nombre de facteurs) traduira une structure minimale de la base de données.
Exemple:soit la relation "facture" suivante:



Ceci met en évidence une dépendance fonctionnelle
DF1= (Projet, atelier, fournisseur, date)➔ montant

Mais aussi en décomposant:
DF2= (projet, date)➔ besoin
DF3=(besoin, atelier)➔ commande
DF4=(commande, fournisseur)➔ montant
DF5= (projet, date, atelier)➔ commande
DF6= (besoin, atelier, fournisseur)➔ montant
Etc…

Dans cet ensemble de DF, il y a des redondances. Comment en extraire le sous ensemble minimum?
Solution:
Posons projet=p, date=d, besoin=b, atelier=a, commande=c, fournisseur=f, montant=m;
Trouver la fonction caractéristique. La simplifier, donner la structure optimale.

BD40 TD N°8: performances

1) Problèmes de recherche d'information.

Exemple1 d'une jonction entre deux tables A de n éléments et B de p éléments: Select * from A inner join B on a=b.

analyser les performances

exemple 2: theta jointure
supposons que A et B n'aient pas un identifiant unique:
Select * from A inner join B on (a=b OR a'=b')
Même question.

2) problème de tri d'un tableau A de dimension n:
algorithme par dichotomie:

a) séparer le tableau en deux sous tableaux: chaque élément doit être comparé avec une valeur médiane, puis déplacé: n couples d'opérations.
b) recommencer *(log_2 n fois)*

Mais les opérations de déplacement ont un coût qui dépend de la taille des champs de A

Clef	infos

On trie suivant une clef de dimension k et on déplace K >> k

Coût total: (K.n)*$\log_2$ n peut vite devenir tres lent !

3) hasch-coding

étant donné une fonction de hachage f(X) portant sur un attribut X d'une table A de dimension N, comment organiser les données. F peut prendre k valeurs possible

TP N° 1 prise en main d'ACCESS

1.1. Les tables

1.1.1. Créer une table "personne" (clef, nom, prénom, date naissance, adresse, téléphone). Observer les divers formats de données (type de données, taille des champs,…) adapter les tailles . Fermer la table, la nommer.

1.1.2. Ouvrir la table en mode saisie; entrer une dizaine de personnes, (faire en sorte qu'il y ait plusieurs personnes de même nom, de même prénom. fermer.

1.2. Requêtes sélection

1.2.1. créer une requête qui affiche tous les champs triés par noms

1.2.2. créer une requête qui affiche les informations d'une personne dont le nom est lu au clavier, puis une requête qui affiche pour un nom et un prénom donné.

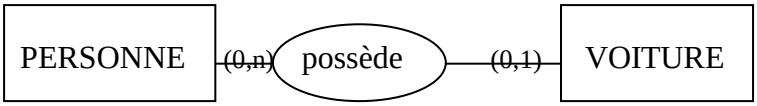
1.2.3. Montrer l'existence du programme SQL correspondant.

1.3. formulaires

1.3.1. en utilisant l'assistant, créer un formulaire de saisie de la table, jouer avec la présentation

1.3.2. faire un formulaire associé à la requête1 sans utiliser l'assistant.

TPN°2: associations



2.1. créer une table voiture (marque, type, immatriculation), la remplir avec une dizaine de voitures.

2.2. Créer la table de l'association "possède", la remplir; (une personne peut posséder une ou plusieurs voiture, une voiture n'appartient qu'à une seule personne. Faire la requête qui affiche les immatriculations des voiture possédées par une personne donnée. Observer ce qui se passe quand on ne met pas les liens graphiquement.

2.2. Observer le texte SQL.

2.3. Ajouter un champ date d'achat dans possède. Faire une requête qui affiche toutes les personnes ayant leur voiture depuis trois ans au moins. Apprendre à manipuler les date

2.4. Requêtes ajout

2.5.1. ajout simple dans la table personne avec type de requête= ajout, qui pose les question du type [NOM?], [PRENOM?] valider les données par un critère, exemple (date de naissance>1/1/99); exécuter. Constater qu'on ne peut entrer qu'un enregistrement à la fois.

2.5.2. Méthode 2: passer par une table temporaire PERSONNE-TMP par exemple, de même format que PERSONNE. Créer un formulaire de saisie pour cette table, (qui peut donc saisir plusieurs enregistrements) avec un bouton "terminé" qui arrête la saisie et ouvre une requête d'ajout qui ajoute tous les champs de PERSONNE_TMP dans PERSONNE. Exécuter deux fois, observer le résultat Comment y remédier puisqu'un bouton ne peut lancer plusieurs opérations à la fois ? ➔ utiliser des macros qui groupent les actions.

2.5.3. Vérifier que les dernieres informations saisies sont bien prises en compte

TP 3. Macro:

3.1.Créer une macro avec deux instructions: lancement de la requête ajout, lancement d'une requête suppression des champs de PERSONNE_TMP. Cette fois le bouton "terminé" lance la macro.

3.2.Travailler la présentation ,(supprimer les demandes de confirmations inutiles dans les requêtes, les affichages de fond, etc...)

3.3.Faire macro qui ouvre le formulaire de saisie, ferme le formulaire, lance la requête d'ajout, lance la requête de suppression.

3.4. Observer les divers types d'actions possibles dans les macro.

3.5. Saisie contrôlée, SQL

2.4.Faire une saisie des personnes avec contrôle que le nom n'existe pas déjà: utiliser intuitivement SQL dans la requête ajout avec une contrainte

(*insert…into…..where Personne_tmp.nom* ***not in*** *select nom from personne)*

Faire de même avec contrainte {nom, prénom} n'existe pas déjà. Constater que:

(*insert…into…..where Personne_tmp.nom not in select nom from personne*

and Personne_tmp.prénom not in select nom from personne EST FAUX, corriger en utilisant EXISTS.

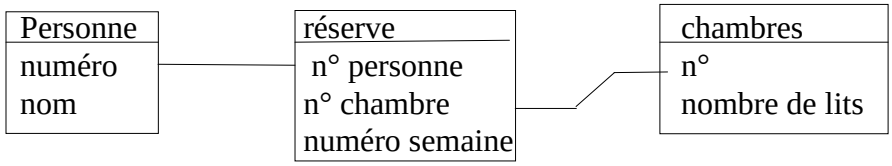
3.6. SQL saisie de la table possede: génération automatique des clefs externes: Faire un formulaire de saisie des noms et des voitures et une macro avec 4 requêtes::

3.6.1. une qui ajoute les noms à personne et génère le n°

3.6.2. idem pour la table des voitures.

3.6.3. une qui ajoute le numéro de personne et le numéro de voiture à la table "possède" .

SQL Base de données des réservations de chambres d'hotel: en ACCESS



1. faire une requête sélection qui affiche les chambres disponibles une semaine donnée (infaisable en mode graphique) pour ceci:

1.1. faire une requête R1 qui affiche les chambres occupées. Observer l'instruction SQL. Faire ressortir le fait que "SELECT DISTINCTROW" définit un ensemble E, et que cet ensemble peut donc intervenir dans une autre instruction.

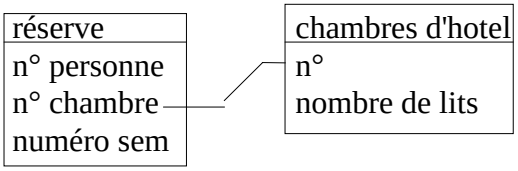
1.2. Faire la requête R2 qui selectionnera les chambres d'hotel "NOT IN E".

a) faire une requête qui sélectionne toutes les chambres de l'hotel

b) passer en SQL et ajouter à l'instruction SELECT la condition WHERE chambres.n° NOT IN . . . .

constater qu'il y a deux versions:

a) en laissant la jointure réserve.n°chambre ⇔ chambre.n°



qui génère une clause INNER JOIN. Constater que c'est faux, trouver la cause.

b) en supprimant la jointure, c'est OK, expliquer

2. on dispose de deux tables *réservations(personne, n° de chambre, date),et annulations(personne, n° de chambre, date),* et d'une table temporaire *annulations_tmp(personne, n° de chambre, date),*ajout à la table des annulations seulement si la réservation a été faite et si l'annulation n'a pas déjà été faite.

BD40 TP4: interfaçage entre ACCESS et l'extérieur

1. importer et exporter des tables crées par un programme quelconque

1.1.importation de table, format fixe ou avec délimiteur

a) créer une table avec 3 champs réels simples
b) - menu ➔"fichier" ➔ "données externes",➔"importer".
c) Idem en utilisant une macro (action "transférer texte")

1.2. exportation de table quelconque vers fichier.

2. lancer une application à l'aide d'une macro

lancer un programme fixe (exemple lancer word sur un document donné)
dans le menu "macro" choisir "exécuter application"; dans le champ " "ligne de commande", taper une ligne de commande DOS quelconque (chemin complet accédant à un .COM, .EXE ou .BAT)
exemple: `c:\….\WORD a:document.doc`

2.1. lancer un programme défini par un formulaire.

a) Créer une table T avec des champs "application" , "disque", 'fichier".
b) La remplir avec des informations du type "C:\….\word "A:" "document15.doc"
d) Créer un formulaire F permettant de sélectionner une ligne de cette table.
e) Faire une macro qui exploite les champs sélectionnés sous la forme d'une concaténation de champs :

Ligne de commande:

```
= [forms]![F]![application] +" " +  [forms]![F]![disque] + [forms]![F]![fichier]
```

indispensable espace indispensable

2.3. utiliser 1 et 2 pour sous traiter à un programme (en C par exemple) ce qu'on ne peut pas faire en SQL: exporter les données dans un fichier F1, appeler le programme C avec F1 en paramêtre, récupérer les résultats F2 (exemple crypter décrypter une table, programme joint)

3. tables EXCEL.
   a) **importe**r une table comme en (1); constater que dans ACCESS et EXCEL on a alors deux tables distinctes: la modification de l'une n'entraîne pas la modification de l'autre.
   b) **attacher** une table EXCEL: "données externes"➔"lier des tables". Observer que cette fois on a une seule table.

4. utiliser les liens hyper-textes
   a) créer une table avec un champ de type "lien hyper texte"
   b) ouvrir un fichier Word comportant des liens hyper texte (exemple polycop c++ fourni) repérer dans ce document les liens (en bleu) sélectionner, copier (^C), et coller dans la table. Constater qu'en cliquant sur le champ on va directement à l'endroit souhaité.
   c) constater que cela peut servir aussi à lancer une application (mais contrairement à(2) on ne peut passer un paramètre.

BD40 TPN°5 Généricité en ACCESS, programmation BASIC

Problème: écrire des requêtes qui puissent s'appliquer à des tables diverses.
Exemple: on a écrit une série de requêtes qui permettent de saisir une association "possède" entre "voiture" et " personne" (TP3). C'est compliqué, on ne va pas tout refaire pour l'association "habite" reliant "personne" et "logement".

Solution: générer la requête en BASIC:

Exercice1:
Faire un formulaire avec seulement un bouton "Exec" auquel on associe la requête SQL:
requete1= select * from client.
Observer le code généré: cliquer sur "exec" (avec bouton droit de la souris), clicker sur "propriété", "code généré" on obtient (à quelques détails près sur le traitement des erreurs):

```
Option Compare Database
Option Explicit
Private Sub Commande0_Click()          <-- commande 2 déclenchée sur click
    Dim stDocName As String            <-- déclaration d'une chaine de caractère
      stDocName = "Requête1"           <-- on traite la requète1
      DoCmd.OpenQuery stDocName, acNormal, acEdit  <-- exécution de la requête

      Exit_Commande1_Click:
      Exit Sub
End  sub
```

au lieu d'exécuter la requète 1 telle qu'elle est, , on peut générer soi même en BASIC du texte SQL : faire un deuxième bouton qui lance l'exécution de la même requète qui sera calculée par le programme suivant

```
Private Sub Commande1_Click()          <-- Commande déclenchée sur click
    Dim chaineSQL As String
    Dim ComSQL As QueryDef             <-- Déclarations
    Dim bd As Database
    Set bd = CurrentDb                 <-- travail dans la base en cours
    Set ComSQL = bd.QueryDefs("Requête1")  <-- On reconstruit la requête 1
    chaineSQL = "SELECT * FROM clients;"   <-- dont le texte est ceci
    ComSQL.SQL = chaineSQL
    Exit Sub
End Sub
```

L'instruction chaineSQL="select * from client" définit la requète "requète1" à exécuter; ici c'est une constante.

Nota
a)cette dernière procédure est automatiquement générée par ACCESS lors de la création du bouton: "propriétés", "sur click", procédure événementielle".

b) Les deux procédures peuvent être fusionnées.

Exercice 2: même chose mais en construisant la requête par concaténation d'information en provenance d'un formulaire:

a) Partir d'une table des tables avec les champs "nom de table" "nom du champ1", "nom du champ2", etc…

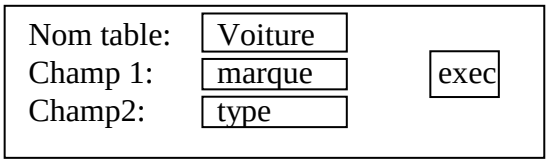
Exemple:

table_des_tables			
Nom_table	champ1	champ 2	etc…
Personne	nom	prénom	
Voiuture	marque	type	immatriculation

b) Faire un formulaire qui sélectionne une ligne de cette table,

Exemple

F1=



c) Construire par concaténation la requête qui sélectionne les champs de cette ligne

Nota: si le nombre de champs de la table est variables, on peut évidemment introduire une concaténation conditionnée par (champ 2 = null)

Exercice 3

Idem avec plusieurs tables, refaire le TP3: construire une requète qui puisse aussi bien saisir les occurrences des tables PERSONNE , POSSEDE, VOITURE que des tables PERSONNE HABITE, LOGEMENT à partir d'une table temporaire TMP

faire une seule requête qui ajoutent à une table quelconque (personne ou voiture) les informations (si elles n'existent pas déjà) en provenance d'une table temporaire TMP.

faire la requête qui ajoute les pointeurs dans l'association "possède" ou toute autre association X entre une entité A et une entité B

BD40: Dernier TP interface homme machine

Faire une base de donnée répondant aux exigences de convivialité et sécurité de l'utilisateur.

Sujet: base de données de l'hôtel
Entités:

client(#code, titre, nom, prénom, adresse, code_postal, ville, date entrée)
chambres(#n°, nombre de lit , type-toilettes, prix)
réservations(N° client, N°chambre, date arrivée prévue , date départ prévue date arrivée réelle , date départ réelle)
facture(#n°facture, code_client, date , payé(o/n))

formulaires:

menu principal, saisie_client, saisie_réservation, saisie annulation, consultations et modifications,

requêtes particulières

saisies contrôlées
établissement de factures à partir des réservations (prévues ou réelles)

états :

facture, statistiques d'occupation par semaine, etc..

REGLES:

a) Amélioration de la saisie:

Utiliser les listes déroulantes pour ne pas ressaisir une chambre connue
Utiliser des sous formulaires pour la saisie des réservations
Gérer les formulaires (un seul ouvert à la fois)
Mises à jour optimisées; exemple: augmenter de 5% les prix des chambres,
Suppression des messages d'avertissements

b) offrir des outils de supervision des ventes (histogrammes des nombres de clients par exemple) et autres outils de prestige.

c) sécurité:

mot de passe
saisie sécurisée (contraintes)
suppression de tous les boutons et icônes inutiles (l'utilisateur ne doit pas pouvoir modifier la structure de la base, par exemple effacer une table)