ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Tableaux dynamiques: *vecteurs*

Pour pallier les défauts inhérents à la rigidité des tableaux de taille fixe (*built-in array*), la librairie (générique) standard[1] de C++ fournit un type de donnée[2] dénommée ***vector*** (*vecteur*), offrant au programmeur un moyen très efficace pour construire des structures de données permettant de représenter des **tableaux de tailles variables** (i.e. *tableaux dynamiques*)[3].

La taille de ces «tableaux» n'est pas obligatoirement prédéfinie, et peut donc varier en cours d'utilisation.

Pour pouvoir utiliser ces *vecteurs* dans un programme, il faut, comme dans le cas des entrées-sorties, importer les prototypes et définitions contenus dans la librairie, au moyen de la directive d'inclusion:

```
#include <vector>
```

1. Le nom officiel de cette librairie est ***STL*** (*Standard Template Library*)
2. En fait de type, il s'agit en réalité d'un *chablon* de *classe* (***template classe***), c'est-à-dire une définition **générique** (valide et réutilisable pour n'importe quel type, de base ou complexe).
3. Pour être exact, les *vectors* sont plus que de simples tableaux dynamiques; ils s'inscrivent dans un famille plus générale d'éléments, utilisés comme briques de bases pour les structures de données complexes, éléments que l'on appelle ***conteneurs*** (*containers*) ou ***collections***, et pour lesquels un ensembe de caractéristiques et contraintes très précises sont définies, comme par exemples les performances minimales des algorithmes d'accès et de recherche.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Vecteur: déclaration (1)

Un *vecteur* peut être déclaré selon la syntaxe suivante:

vector<«*type*»> «*identificateur*»;

Avec *type* n'importe quel type, élémentaire ou non, et correspondant au type de base du tableau.

Exemple:

```
#include <vector>
...
vector<int> age;
```

Il s'agit d'une déclaration de variable («age») tout à fait traditionnelle, dans laquelle la séquence «vector<int>» correspond à l'indication du type de la variable, en l'occurence un tableau dynamique (*vecteur*) d'entiers.

On voit dans ce cas clairement ressortir la nature composite du type.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Vecteur: déclaration (2)

Le fait que l'on s'intéresse ici à des *collections* d'un nombre potentiellement variable d'éléments explique que la déclaration puisse ne comporter aucune indication sur la taille initiale du tableau. Une variable ainsi déclarée correspond alors tout simplement à un tableau vide.

Cependant, une taille initiale peut, si nécessaire, être indiquée; la syntaxe de la déclaration est alors:

```
vector<«type»> «identificateur»(«taille»);
```

Un tableau nommé `identificateur` comportant `taille` éléments de type `type` sera créé, chacun des éléments ayant comme valeur la valeur par défaut de `type` généralement une valeur dérivée de l'expression `(0)`.

Exemple:

```
vector<int> age(5);
```

Correspond à la déclaration d'un tableau d'entiers, initialement composé de 5 éléments valant 0.

	age
age[0]	0
age[1]	0
age[2]	0
age[3]	0
age[4]	0

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Vecteur: déclaration avec initialisation

La déclaration d'un ***vecteur*** peut être associée à une **initialisation explicite** des éléments initiaux; cependant, cette initialisation ne pourra consister qu'en (a) une duplication d'un même élément, ou (b) en une duplication d'un *vecteur* pré-existant:[4]

(a) `vector<«type»> «identificateur»(«taille»,«valeur»);`
où *valeur* est une expression de type *type*, dont le résultat sera pris comme valeur intiale des *taille* éléments du tableau *identificateur*.

Exemple:

```
vector<int> vect1(5,8);
```

déclare le *vecteur* d'entiers `vect1` avec un contenu initial de 5 entiers valant «8»

(b) `vector<«type»> «identificateur»(«id-vector»);`
où *id-vector* est un identificateur de *vecteur* de type de base *type*.

Exemple:

```
vector<int> vect2(vect1);
```

déclare le *vecteur* d'entiers `vect2`, avec un contenu initial identique au contenu de `vect1` (duplication).

4. Contrairement au cas des tableaux de taille fixe, il n'existe pas de moyen simple pour exprimer la valeur littérale d'un *vecteur* dont les éléments n'ont pas tous la même valeur.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Vecteur: constante

Comme pour tous les autres types, il est possible de déclarer des **constantes** de type *vecteur*

La syntaxe est identique à celle des autres déclarations de constantes:

```
const vector<«type»> «identificateur»(«initialisation»);
```

identificateur correspond alors à un *vecteur* dont tant le nombre d'éléments que la valeur de chacun de ces éléments sont fixes (et ne peuvent donc êtres modifiés).

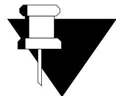
Exemple:

```
const vector<int> age;
```

Correspond à la déclaration d'un *vecteur* constant vide (ne contenant aucun élément) et auquel aucun élément ne pourra être ajouté[5]

```
const vector<int> vect2(vect1);
```

Correspond à la déclaration d'une copie figée (*snapshot*) du vecteur `vect1`.



Il n'est pas possible de déclarer des vecteurs de constantes.
Ainsi, la syntaxe `vector<const «type»> «identificateur»` bien que licite en soit, n'est en pratique pas utilisable.

5. Cette déclaration est bien sûr totalement inutile.

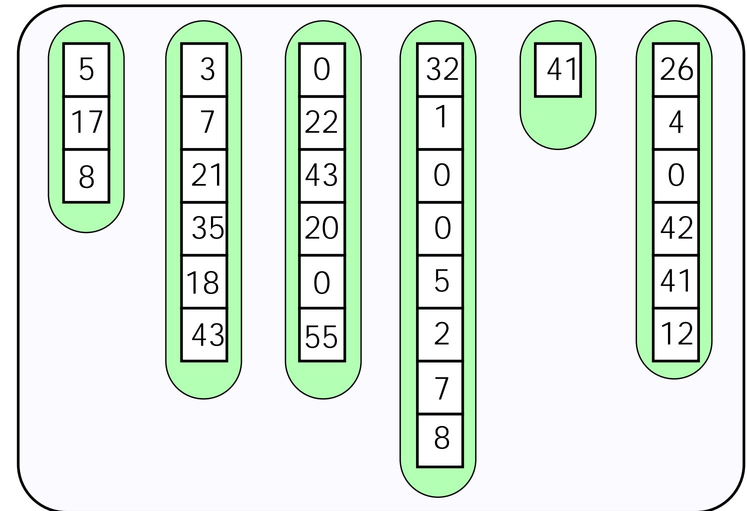
Vecteur: vecteur de vecteur

Le type de base d'un *vecteur* est peut être un type quelconque, y compris composé. En particulier, le type de base d'un *vecteur* peut être lui-même un *vecteur*.

```
vector<vector<int>  > matrice;
```

Attention, dans le cas d'une déclaration directe d'une telle structure, il est obligatoire de séparer les «>», présents en fin de définition du type, par un espace[6].

D'un point de vue sémantique, les *vecteurs* de *vecteurs* ne correspondent pas (nécessairement) à des matrices, mais simplement à des ensembles d'ensembles d'éléments.



vector<vector<int> >

6. Cette contrainte est en fait une convention adoptée pour distinguer ce typage de l'opérateur «>>»

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Vecteur: affectation

Toute **variable**[7] de type *vector* peut être modifiée (globalement) par affectation[8]:

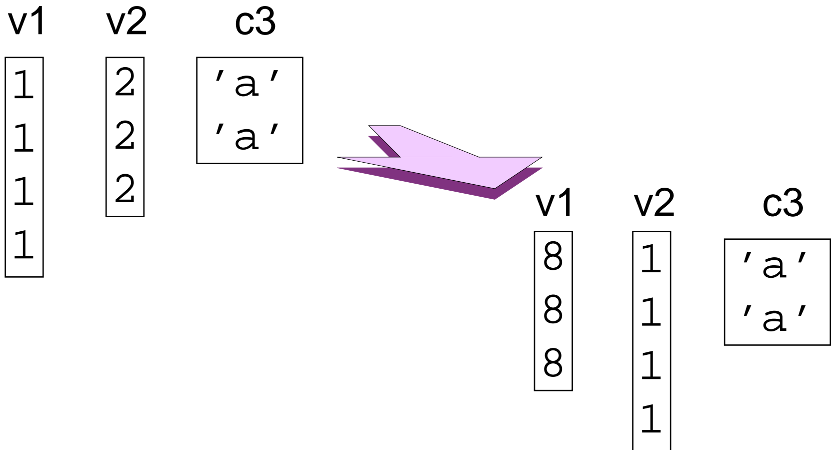
```
«identificateur» = «valeur»;
```

où *valeur* est une expression de même type qu'*identificateur*, notamment en ce qui concerne le type de base.

Dans ce cas, la modification va porter sur l'ensemble des éléments de la structure:

Exemple:

```
// 1) Déclarations-initialisations
vector<int> v1(4,1);
vector<int> v2(3,2);
vector<char> c3(2,'a');
// 2) Affectations .........
v2 = v1;
v1 = vector<int>(3,8);
c3 = vector<int>(4,7); // ILLEGAL
```



7. Sauf [naturellement] les constantes.
8. Parmis les opérateurs définis pour les *vecteurs*, on trouve en effet celui d'*affectation*, i.e. « = ».

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Vecteur: accès aux éléments

Chaque élément d'un vecteur est **accessible individuellement**.

Différents moyens permettent d'accéder aux éléments,
et notamment l'indexation comme dans le cas des tableaux de taille fixe:

L'***indice***[9], **placé entre crochets** «[ ]», indique le rang de l'élément dans le tableau.

Les éléments d'un vecteur de taille *n* sont sont numérotés de 0 à *n*-1.
Dans le cas d'accès via l'opérateur «[ ]»,
Il n'y a pas de contrôle de débordement du tableau !

9. Indice qui peut être une expression numérique.

Vecteur: opérateurs relationnels

Les **opérateurs relationnels** suivant
sont définis pour les vecteurs:

Opérateur	Opération	
<	strictement inférieur	comparaison lexicographique des éléments.
<=	inférieur ou égal	
>	strictement supérieur	
>=	supérieur ou égal	
==	égalité	
!=	différence (non-égalité)	

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Vecteur: méthodes (1)

Un certain nombre d'opérations,
directement liées à l'aspect «ensembliste» des vecteurs,
sont définies dans la librairie *STL*.

L'utilisation de ces opérations particulières, appelées ***méthodes***[10],
se fait par le biais de la syntaxe suivante:

```
«id-vector».«id-methode»(«arguments»);
```

Exemple:

```
individus.puch_back(jean);
```

à comme effet d'appliquer la méthode `push_back`, au vecteur `individus`, en prenant `jean` comme paramètre [de la méthode].

Comme les fonction, les méthodes peuvent éventuellement retourner une valeur.

10. Les méthodes sont des éléments informatiques (séquences d'instructions) issus de l'extension «objet» de C++. Pour le moment, il vous suffit de les considérer comme des fonctions ayant une syntaxe d'appel un peu particulière.

EPFL
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Vecteur: méthodes prédicats

Parmi les méthodes disponibles, on trouve[11]:

Prédicats:

- `int size()`: renvoie la taille du *vecteur* (i.e. son nombre d'éléments).

  Une manière usuelle pour parcourir les éléments d'un *vecteur* est donc l'itération `for` suivante:

```
for (int i(0); i<vect.size(); ++i)[12]
{
   // traitements
}
```

- `bool empty()`: indique si le vecteur est vide (ou non).

  On a l'équivalence suivante: «`vect.empty()`» <=> «`(vect.size() == 0)`»

11. Par convention:
    - «`vect`» désignera la variable *vecteur* pour laquelle la méthode est invoquée;
    - «`base-type`» désignera le type de base d'un *vecteur*

12. Dans le cas où le sens de parcours est sans importance, on pourra avantageusement remplacer cette séquence par cette autre, plus efficace:
    `for (int i(vect.size()-1); i >= 0; --i) { ... }`

EPFL
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Vecteur: méthodes de mise à jour

Modificateurs:

- `void clear()`: vide le vecteur, en supprimant tous ses éléments.
  Après l'invocation de cette méthode, le prédicat «`empty`» est forcément vrai.
- `void pop_back()`: supprime le dernier élément du vecteur.
- `void push_back(const base-type element)`: ajoute `element` à la fin du vecteur; `element` devient donc le nouveau dernier élément du vecteur.

Exemple: La boucle suivante initialise un vecteur d'entiers positifs, de taille 8, en demandant à l'utilisateur de saisir les valeurs initiales.
Lors de la saisie, l'utilisateur a la possibilité d'effacer la valeur précedemment saisie, en indiquant un chiffre négatif, ou d'effacer tout le vecteur, en entrant 0.

```
while (vect.size() < 8) {
   int val;
   cout << "Entrez coefficient " << vect.size() << ':' << flush;
   cin >> val;
   if (val < 0)  {vect.pop_back(); continue;}
   if (val == 0) {vect.clear(); continue;}
   vect.push_back(val);
}
```

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Vecteur: méthodes d'accès

Accès:

- base-type **front**( ): renvoie une référence vers le premier élément du vecteur
  Les séquences «vect.front()» et «vect[0]» sont donc équivalentes.
  Précondition: le vecteur n'est pas vide (le prédicat «empty()» est faux).
  Exemple: L'itération suivante a comme effet de diviser tous les éléments d'un vecteur (à l'exception du premier), par le premier élément du vecteur.

```
for (int i(1); i<vect.size(); ++i)
{vect[i] /= vect.front();}
```

- base-type **back**( ): renvoie une référence vers le dernier élément du vecteur
  Les séquences «vect.back()» et «vect[vect.size()-1]» sont équivalentes.
  Précondition: le vecteur n'est pas vide (le prédicat «empty()» est faux).
  Exemple: remplir un vecteur d'entiers, en demandant chaque valeur à l'utilisateur; le vecteur est considéré comme plein lorsque l'utilisateur entre la valeur 0.

```
do { vect.push_back(saisirEntier)
} while (vect.back());
```

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Commande `typedef`

L'utilisation de variables dans un programme implique généralement que l'on spécifie plusieurs fois le type de ces variables (lors de la déclaration des variables, lors du prototypage des fonctions qui en font usage, lors de conversions, etc...)

Lorsque le type est complexe, sa définition peut être ardue, et est généralement longue, ce qui ne facilite pas la lecture du programme. En outre, les modifications éventuelles à apporter à la définition du type doivent être opérées sur chacune de ses occurences. Comme dans le cas des «blocs d'instructions réutilisables» (i.e. les fonctions), et pour les mêmes raisons, la duplication de la définition d'un type est à éviter.

La commande `typedef` permet pour cela de définir des *synonyme* (***alias***) de types, qu'ils soient fondamentaux ou composés:

```
typedef <type> <alias>;
```

Cette instruction permettra de désigner le type `type` indifféremment par `type`, ou au moyen de l'identificateur `alias` (`typedef` n'introduit pas de nouveau type, mais un nouveau nom pour le type)

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Commande `typedef`: exemple

```
typedef int longueur;
longeur diametre, rayon;
int nbCercles;
...
void  traceCercles(longueur d,
      longueur r, int nombre)
{ ... }
```

```
// pas de typdef //
int diametre, rayon;
int nbCercles;
...
void  traceCercles(int d,
      int r, int nombre)
{ ... }
```

Dans ces deux exemples, `diametre`, `rayon` et `nbCercles` sont tous de même type (`int`). Mais, si l'on utilise `longueur` pour exprimer toutes les longeurs (programme de gauche), et que pour une raison quelconque on est amené à changer la représentations des longueurs (p.ex. par des réels), il suffira d'opérer ce changement dans la définition de l'alias `longueur`, plutôt qu'à chaque occurence de `int` représentant une longueur.

```
typedef float longueur;
longeur diametre, rayon;
int nbCercles;
...
void  traceCercles(longueur d,
      longueur r, int nombre)
{ ... }
```

```
// pas de typdef //
float diametre, rayon;
int nbCercles;
...
void  traceCercles(float d,
      float r, int nombre)
{ ... }
```

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

typedef et types composés

Comme dans pratiquement tous les cas de types composés, la commande «**typedef**» permet de simplifier les déclarations (plus de lisibilité).

Par exemple dans la cas des tableaux (*built-in* ou *vecteurs*), en fournissant un alias pour le couple (*indication de tableau*, type de base):

typedef «*type-base*» «*alias*»**[**«*taille*»**]**

typedef vector<«*type-base*»**>** «*alias*»

Dans le cas des tableaux multidimensionnels, on peut se servir avantageusement de cette commande pour rendre plus explicite les déclarations et les usages ultérieurs:

```
typedef int Vecteur2[2];             // définis le type Vecteur2 comme un tableau de 2 entiers
typedef vector<int> GrilleLoto;      // et GrilleLoto comme un ensemble d'entiers
typedef Vecteur2 Matrice3x2[3];      // définit le type Matrice3x2 comme un tableau de 3 Vecteur2
typedef vector<GrilleLoto> BulletinLoto; // et BulletinLoto un ensemble de GrilleLoto
Matrice3x2 A = {{1,2},{3,4},{5,6}}
BulletinLoto fortune;
```

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Création du schéma relationnel (1)

La démarche générale est:

1. **Traduction du schéma *entité-association* en un schéma *relationnel***,
   en utilisant un algorithme de traduction, et les règles de modélisation précédentes.

2. ***Amélioration* éventuelle du schéma relationnel** ainsi obtenu,
   par une décomposition en relations en *troisième forme normale*
   (sans perte d'information ou de dépendances fonctionnelles).

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Création du schéma relationnel (2)

L'algorithme de traduction est le suivant[1]:

(a) Pour chaque TE, créer une relation:

- dont le **nom** est le nom du TE
- dont les **attributs** sont les attributs monovalués du TE avec, comme nom, la concatenation du nom du TE et du nom de l'attribut du TE (par exemple, l'attribut `matricule` du TE `Etudiant` sera nommé `Etudiant.matricule`)
- dont **l'identifiant** est constitué des attributs identifiants du TE (si tous les identifiants sont multivalués, alors il faut créer un attribut identifiant spécifique supplémentaire pour la relation).

Remarque:

la relation de nom `R` décrite par les attributs `R.X1, ..., R.Xn` pourra être notée plus simplement `R(X1,...,Xn)`.

1. Pour simplifier, on ne considère ici que des attributs simples. les éventuels attributs complexes sont à modéliser en respectant les règles énoncées précédemment.

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Création du schéma relationnel (3)

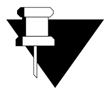
(b) Pour chaque TA, créer une relation:

- dont le **nom** est le nom du TA
- dont les **attributs** sont:
  - les attributs monovalués du TA avec, comme nom, la concatenation du nom du TA et du nom de l'attribut (par exemple, l'attribut `salle` de la relation `Suivre` sera nommé `Suivre.salle`)
  - les attributs identifiants des TE liés au TA avec, comme nom, la concatenation du nom du TE, du rôle du TE et du nom de l'attribut
- dont **l'identifiant** est constitué des attributs identifiants du TA.
- dont les **identifiants externes** sont les identifiants de TE liés (qui référencent les relations décrivant ces TE).

Création du schéma relationnel (4)

(c) Pour chaque attribut multivalué d'un objet ○ (TE ou TA), créer une relation

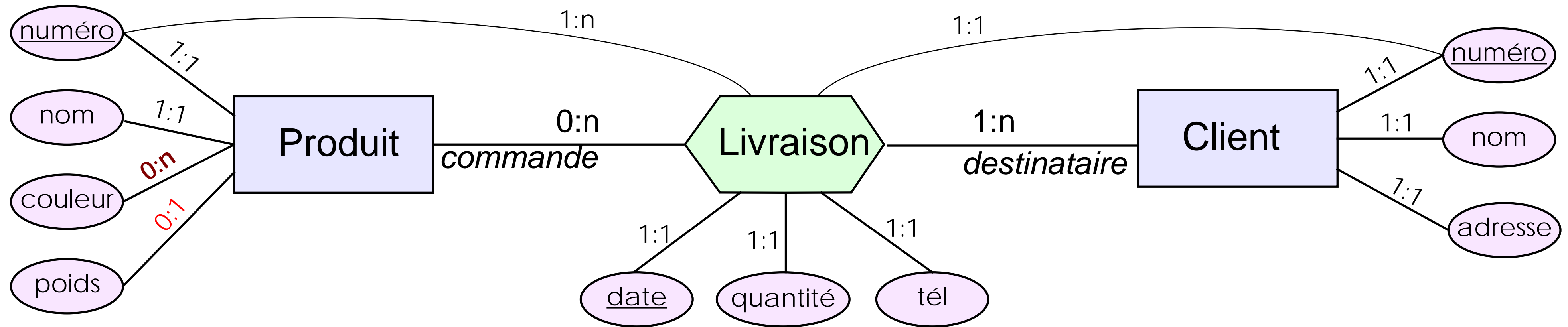
- dont le **nom** est le nom de l'objet ○ concaténé à celui de l'attribut
- dont les **attributs** sont l'attrbut lui-meme et les attributs identifiants de l'objet ○
- dont **l'identifiant** est constitués de l'attribut et des attributs identifiants de l'objet ○
- dont les **identifiants externes** sont les identifiants de de l'objet ○ (qui référencent la relations décrivant cet objet).



Les domaines de valeurs des attributs des relations sont les domaines de valeurs des attributs de TE ou de TA correspondants (plus, sauf pour les attributs des relation correspondant à des attributs multivalués, une ***valeur nulle*** **«*»** si l'attribut d'origine est **facultatif**)

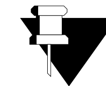
Création du schéma relationnel (5)

Soit le schéma relationnel suivant:



En appliquant l'algorithme de traduction précédent, on obtient:[2]

```
(a)   Produit(numéro, nom, poids(*))
(c)   Produit.Couleur(couleur, Produit.numéro)
(a)   Client(numéro, nom, adresse)
(b)   Livraison(date, quantité, tél,
            Produit.commande.numéro,
            Client.destinataire.numéro)
```



«(*)» indique que l'attribut peut avoir une valeur nulle.

2. Afin de ne pas trop alourdir les définitions, les préfixes des noms d'attributs concaténés ne sont indiqué que si le préfixe diffère du nom de la relation.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Amélioration des schémas relationnels (1)

Considérons le schéma relationnel suivant,
décrivant les produits, les clients et les livraisons d'une entreprise :

```
Produit1(numéro, nom, couleur)
Client(numéro, nom, adresse)
Livraison(date, quantité, Client.destinataire.tél,
      Produit.commande.numéro,
      Client.destinataire.numéro)
Produit2(numéro, poids)
```

Ce schéma pose plusieurs problèmes:

- s'il n'y a plus de livraisons pour un client, son numéro de téléphone est perdu;
- il faut ressaisir le numéro de téléphone du client à chaque livraison et, de plus, vérifier s'il est cohérent avec l'information déjà saisie pour les autres livraisons;
- l'information concernant les produits et les clients est éparpilée dans plusieurs relations.

Le schéma n'est donc pas optimal et doit donc être **amélioré**.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Amélioration des schémas relationnels (2)

Une possibilité est:

```
Produit(numéro, nom, couleur, poids)
Client(numéro, nom, adresse, tel)
Livraison2(date, quantité,
      Produit.commande.numéro,
      Client.destinataire.numéro)
```

Ce schéma ne pose plus les problèmes mentionnés.
Il est donc meilleur que le précédent.

D'une façon générale,
les relations qui ne posent pas de problèmes
lors de l'insertion/modification/suppression des tuples
sont appelées des ***relations normalisées*** et le processus général
d'amélioration d'une relation ou de tout schéma relationnel
est appelé ***normalisation***.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Normalisation de schémas relationnels

La normalisation des relations peut être faite en **découpant les relations posant problème** (i.e. *non normalisées*) en **plusieurs relations *mieux formées*** (i.e. *normalisées*) et décrivant la même information.

La normalisation d'un schéma correspondra alors à une décompositions des relations non normalisées en relations normalisées, suivie d'une recomposition des relations normalisées ainsi obtenues, permettant un meilleur regroupement des informations reliées.

Bien sûr, toutes ces transformations devront se faire **sans perte d'information**.

Il existe plusieurs méthodes pour normaliser des schémas relationnels[3] et pour décrire un exemple de telle méthode, nous allons avoir besoin des concepts de ***dépendance fonctionnelle*** et de ***graphe fonctionnel***.

3. Notez cependant qu'aucune de ces méthodes, si elle est appliquée de façon purement automatique, n'est totalement satisfaisante, car on ne peut garantir que les relations normalisées produites seront sémantiquement signifiantes.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Dépendance fonctionnelle (1)

Etant donnée une relation **R** et *X* et *Y* deux attributs (ou ensembles d'attributs) de **R**, on dit qu'il existe une ***dépendance fonctionnelle*** (ou DF) de *X* vers *Y* si la propriété suivante est vérifiée :

> Si deux tuples quelconques de **R** ont les mêmes valeurs pour *X*, alors ils ont aussi nécessairement les mêmes valeurs pour *Y*

La dépendance fonctionnelle de *X* vers *Y* est notée: X→Y.
X (repectivement *Y*) est appelé la ***source*** (resp. la ***cible***) de la DF.

Exemple

Soit une description de produit manufacturé en terme de:
(1) type d'alimentation, (2) voltage, (3) norme de securité et (4) couleur.

alim	volts	norme	couleur
non élec.	0	CE-010	bleu
pile	9	CE-125	bleu
non élec.	0	CE-010	rouge
secteur	230	CE-130	rouge
batterie	9	CE-125	bleu

Dépendances fonctionnelles *sémantiques*

alim→volts
alim→norme
alim→volts,norme

Dépendances fonctionnelles *issues des données*

volts,norme→alim

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Dépendance fonctionnelle (2)

Si Y est réduit à un attribut unique et X est un ensemble minimal d'attributs pour **R** (i.e. $X = x_1, \ldots, x_k$ et il n'existe pas de sous-ensemble strict X' des x_i tel que $X' \rightarrow Y$), la dépendance fonctionnelle est dite **élémentaire**.

(1)	NoProduit → CouleurProduit
(2)	NoProduit → CouleurProduit, PoidsProduit
(3)	NoProduit,CouleurProduit → PoidsProduit

Dans l'exemple ci-contre, (1) est **élémentaire**, tandis que (2) et (3) ne le sont pas.

EPFL ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Notion de graphe minimal des DF

Dépendance fonctionnelle déduite

Si, dans une relation **R**, on a les DF $X \rightarrow Y$ et $Y \rightarrow Z$, alors on a aussi la DF $X \rightarrow Z$ qui est dite DF ***déduite*** (des deux autres).

Une méthode pour déterminer si une DF $X \rightarrow Y$ d'une relation **R** est déduite est de vérifier si, après avoir supprimé la DF $X \rightarrow Y$ de l'ensemble des DF de **R**, Y peut encore être déduite de X.

Graphe minimal des dépendances fonctionnelles

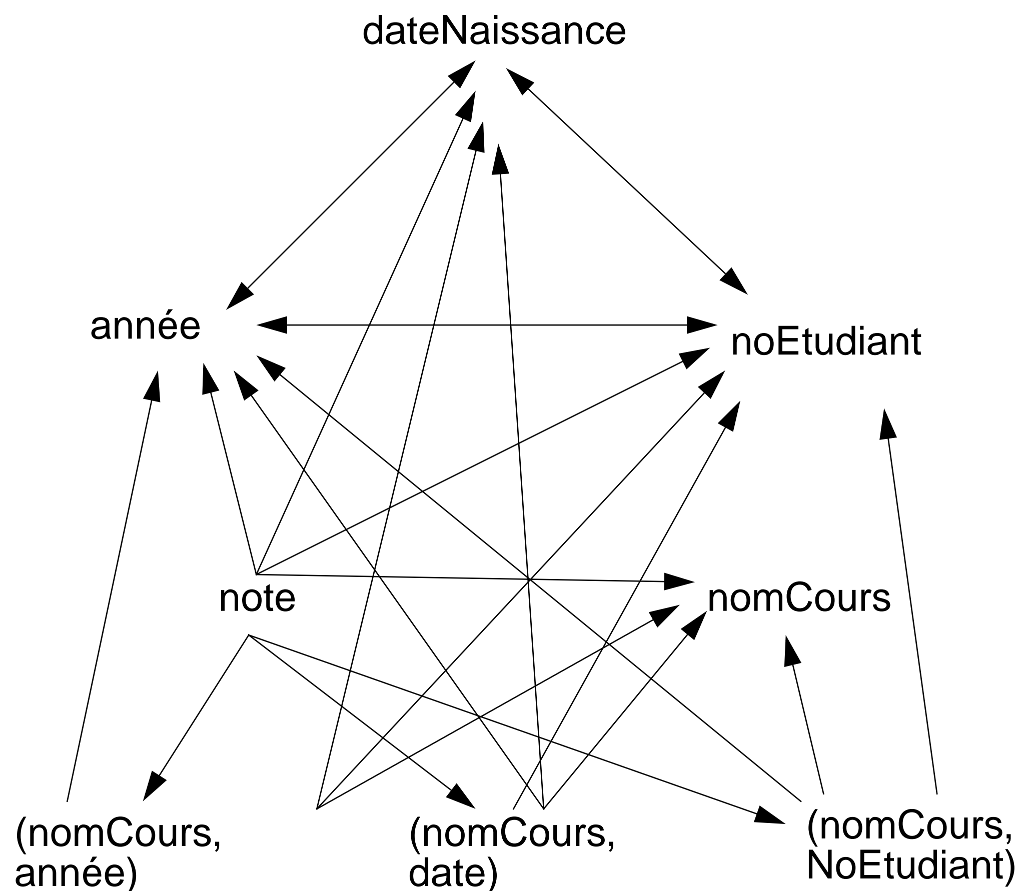
On appelle ***graphe minimal*** (des dépendances fonctionnelles) d'une relation **R** tout ensemble de DF élémentaires

- non déduites,
- dont toute DF élémentaire de **R** peut être déduite.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Exemple de graphe minimal des DF

Graphe complet



dateNaissance	noEtudiant	nomCours	note	année
3/5/59	22	algo	12	1988
3/5/59	22	C	13	1988
2/2/75	41	algo	10	1997

Graphe minimal