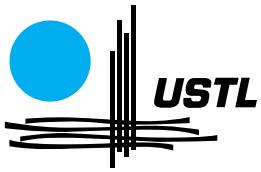


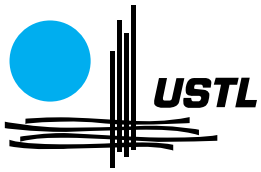

# Cours de Bases de Données

**Jean-Claude Marti**

**1997**

**Correction 2007**

UNIVERSITE DES SCIENCES
ET TECHNOLOGIES DE LILLE




# Cours de Bases de Données

**Jean-Claude Marti**

**1997**

**Correction 2007**

UNIVERSITE DES SCIENCES
ET TECHNOLOGIES DE LILLE

# Introduction

## 1 Qu'est-ce qu'un SGBD

Un SGBD est un système au sein du système, et dans certains cas, ses fonctionnalités (gestion des disques) se substituent à celles du système gérant la machine sur laquelle il est implanté.

C'est un progiciel de stockage et d'exploitation de l'information qui en assure la recherche et la maintenance. Les données sont persistantes (gestion de disques), partagées entre de nombreux utilisateurs ayant des besoins différents, qui les manipulent à l'aide de langages appropriés (graphiques ou "proches" du langages naturel). Le système assure également la gestion de la sécurité et des conflits d'accès (gestion des transactions). Son administration est centralisée (action d'un Administrateur, le DBA). Les SGBD sont munis d'un langage de requêtes et leur conception est établie à partir de trois couches indépendantes : la couche conceptuelle, la couche logique et la couche physique (implémentation).

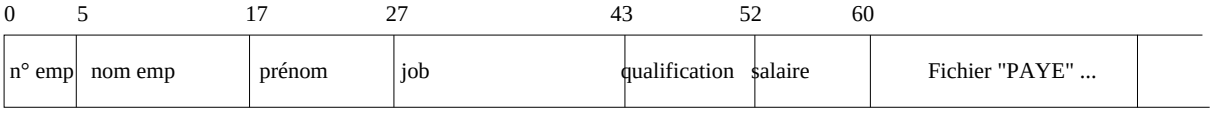
Il faut remarquer que les données sont accessibles directement, alors que les systèmes de banques de données antérieurs ne fournissaient qu'un accès à un ensemble plus ou moins vaste au sein duquel il fallait encore faire une recherche séquentielle. On retrouve ce dernier mode de fonctionnement quand on utilise sur Internet des moteurs de recherche dont la conception a reposé au départ sur des moteurs de bases de données.

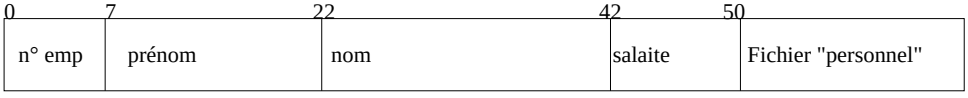
## 2 Historique

Le mot `Data Base` est apparu en 1964 lors d'une conférence sur ce thème aux USA, organisée dans le cadre du programme spatial américain.

Auparavant, on ne connaissait que des systèmes de gestion de fichiers (SGF), basés sur la gestion de bandes magnétiques, destinés à optimiser les accès séquentiels. Les disques étaient alors chers et réservés à de petits fichiers.

La figure suivante montre comment, au sein d'une même entreprise, on pouvait concevoir deux fichiers COBOL permettant la gestion du personnel. De tels fichiers ne pouvaient s'échanger de données sans des programmes d'extraction et de conversion appropriés. Les informations sont en compte d'octet. Le concept de type de donnée au sens actuel du terme n'est pas encore apparu : tout est caractère (octet), même les chiffres des nombres !



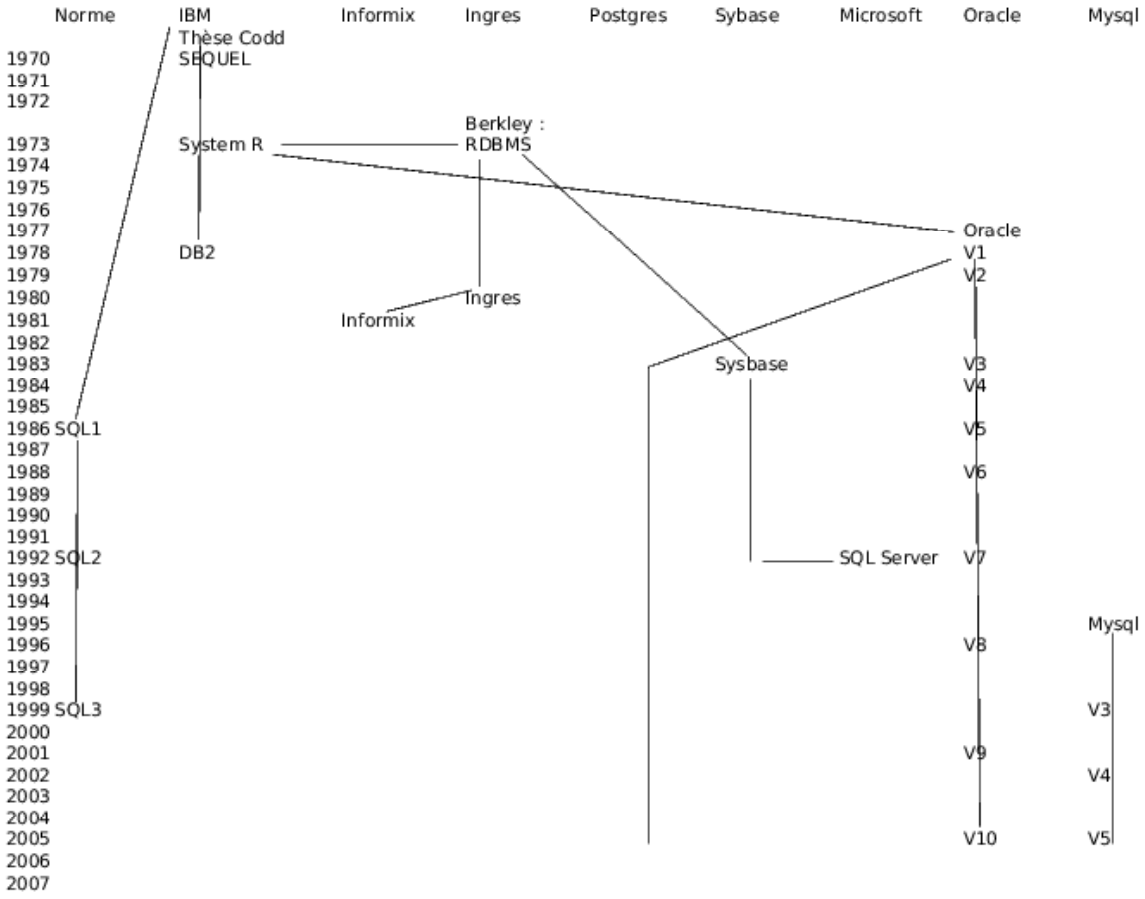

La course à la lune change la donne. Il faut impérativement définir de nouveaux outils, plus efficaces et mieux normalisés. Vers la fin de la décennie 60 apparaissent les premiers SGBD, conçus selon les modèles hiérarchiques, puis réseaux dans la décennie suivante. On voit apparaître des langages navigationnels, inspirés du cobol, et la description des données est indépendante des programmes d'application. Cette première génération suit les recommandations du DTBG CODASYL (Data Base Task Group - Conference On Data System Language), influencé par le système IMS d'IBM.

Le modèle relationnel voit le jour en 70 et met 20 ans pour s'imposer sur le marché. Ce modèle permet la naissance de langages assertionnels, basés sur la logique du premier ordre et les traitements ensemblistes. Dans le même temps, l'emploi des disques se généralise, les accès directs deviennent la règle, le développement des techniques d'optimisation assurent aux SGBD des performances largement équivalentes à celles des anciens modèles de données.

Au cours des années 80, de nouveaux besoins se font jour. Les systèmes mis jusque là sur le marché

privilégiaient des données de gestion. On cherche de plus en plus à manipuler des données techniques, des images, du son. De nombreux travaux de recherche tentent de faire le lien avec le monde *Orienté-Objet* ainsi qu'avec les systèmes d'inférence utilisés en *Intelligence Artificielle*. Compte tenu de l'inertie du marché, il faudra attendre encore une dizaine d'années pour qu'un modèle vraiment nouveau et performant commence à l'envahir. C'est à la fin des années 90 qu'on voit une évolution vers le modèle relationnel-Objet. Aujourd'hui, tous les SGBD qui sortent suivent ce dernier modèle, bien que la majorité des utilisateurs continue de n'en utiliser que la couche relationnelle.

La figure suivante trace l'historique et la filiation des principaux produits.




# 3 Objectifs des SGBD

## 3.1 Indépendance des données

Un aspect important des SGBD vient du principe de découpage en couches, lequel est inspiré du dévelopement d'Arpanet apparu à la fin des années 50, qui assure l'indépendance des données par rapport au système et à l'architecture, mais aussi par rapport aux problèmes du monde réel qu'il s'agit de modéliser.

Le *schéma conceptuel* est issu des travaux des analystes, dirigés par un chef de projet et correspond à une modélisation purement intellectuelle de l'univers à gérer et des paramètres qui doivent être pris en compte. C'est à ce niveau qu'opère le chef de projet chargé de la conception de la base.

Le *schéma logique* en est la traduction informatique abstraite (au sens où on parle de type abstrait de données en algorithmique). Selon le modèle utilisé, les données sont organisées selon un schéma arborescent (modèle hiérarchique), un graphe (modèle réseau), des tables (modèle relationnel), des objets (modèle

objet). A ce niveau travaillent des analystes programmeurs et le DBA (Data Base Administrator) ou administrateur de la base de données. C'est lui qui est responsable de la modification du schéma d'analyse dans le but d'optimiser l'exploitation de la base.

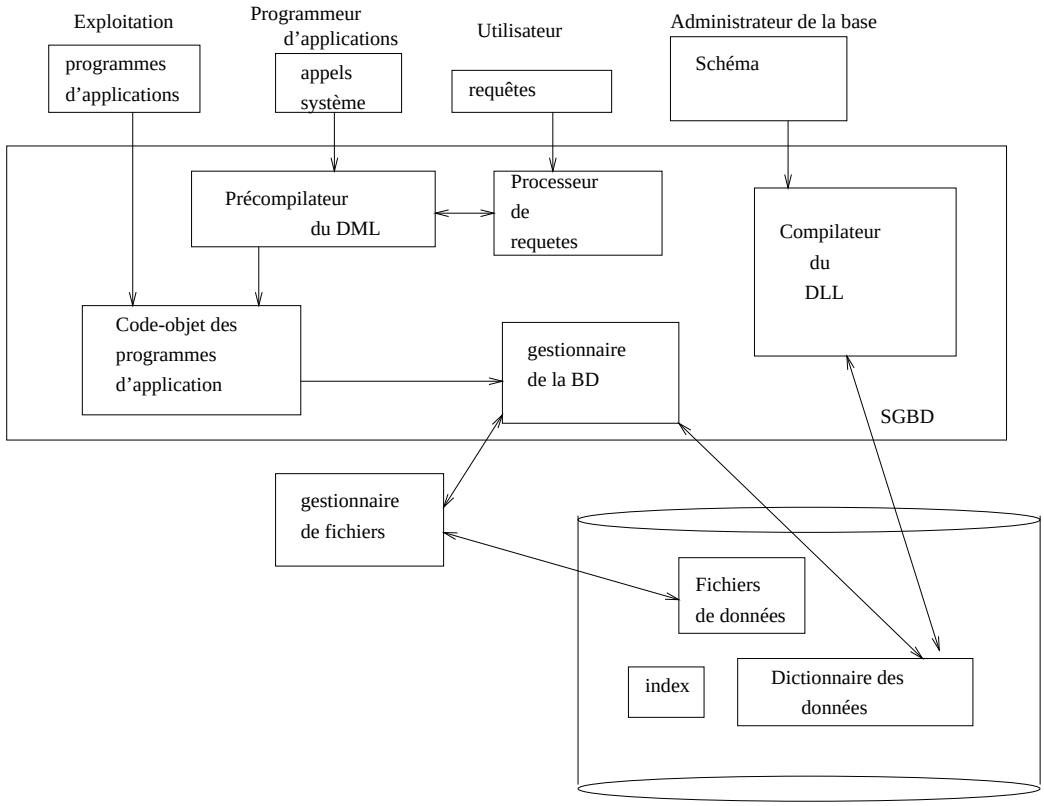
Le *schéma physique* correspond à l'implémentation des données sous forme de fichiers sur disques et gère les mécanismes d'accès. C'est le domaine des programmeurs système, mais aussi celui du DBA qui peut, lorsque le SGBD le permet, décider des modalités d'implantation de la base (position médiane sur le disque, nature des fichiers - séquentiels, ordonnés, hachés, ... ).

Ces 3 niveaux doivent être indépendants, de façon à permettre la modification de l'un sans pour autant remettre en cause la totalité des autres couches, du moins tant que les règles d'interface restent respectées. Les logiciels d'aide à l'analyse permettent tous une transformation automatique et normalisée du schéma d'analyse en un script de création de la base.

## 3.2 Administration centralisée

Elle prend tout son sens dans le contexte historique de développement de l'informatique, par rapport à une époque où, l'anarchie étant la règle, on nageait, au sein d'une même entreprise, dans la plus totale incohérence. Avec un SGBD, personne ne peut modifier de données, encore moins leur mode d'organisation, sans y avoir été dûment autorisé par l'administrateur. Ce dernier a tous les droits sur la base, la crée, l'optimise, suit son évolution, gère les sauvegardes, ajuste les paramètres, distribue les droits d'accès. Il joue le rôle du responsable système vis à vis de ce système spécifique qu'est le SGBD.

Il utilise un *langage de définitions des données* ou DDL en anglais (Data Definition Language), un *langage de manipulation de données* ou DML (Data Manipulation Language), ainsi qu'un *langage de sécurité des données* ou DSL (Data Security Language).




## 3.3 Normalisation

C'est un aspect fondamental qui permet l'échange et la recherche de données à travers des systèmes différents. Malgré les développements particuliers de chaque éditeur de logiciel, la normalisation s'est imposée très tôt, que ce soit au niveau des modèles (ANSI/SPARC, CODASYL) ou des langages (SQL).

Aujourd'hui, la communication à travers des SGBD différents est largement utilisée.

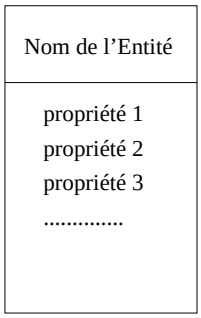
# Le Modèle Conceptuel des Données

Il existe plusieurs systèmes de modélisation. Le plus répandu demeure le modèle *Entité-Association* qui est l'aboutissement de méthodes d'analyse dont la plus utilisée en France est la méthode *MERISE*.

## 1 Concepts

### 1.1 Type d'entité

Une entité ou individu est un objet discernable des autres objets du monde à modéliser, par exemple une personne, un véhicule. Ce peut être aussi un concept ou une grandeur abstraite. Une entité ne peut exister par elle-même sans être déterminée par la liste de ses propriétés ou attributs. Une propriété constitue le plus petit élément d'information ayant un sens intrinsèque : un nom, une date de naissance, un numéro de téléphone, ...



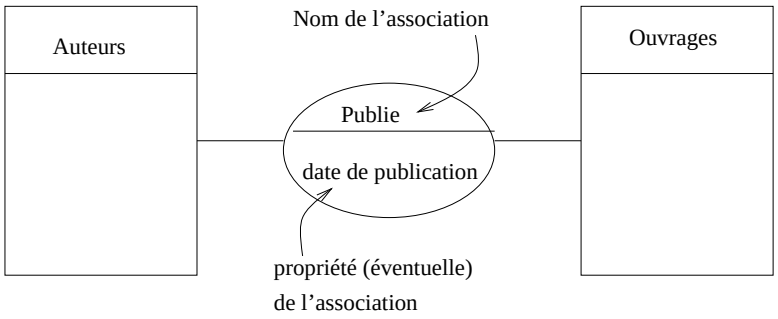

Un type d'entité est représentée par l'ensemble des entités de même nature que l'on pourra déterminer au cours du temps : l'ensemble des individus, des véhicules, des comptes en banque, ...

Chaque entité devra pouvoir être distinguée de façon unique par un *identifiant* ou *clé*. Ce dernier est en général représenté par une propriété particulière. L'usage veut qu'elle soit soulignée dans la représentation graphique du type d'entité.

### 1.2 Association

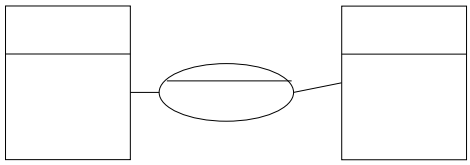
C'est un lien logique entre 2 ou plusieurs types d'entité. Elle est le plus souvent perçue comme une action entre des catégories d'objets et se traduit presque toujours par un verbe. On dira ainsi qu'un auteur *publie* (cf. fig) des ouvrages

Une association peut aussi posséder des propriétés qui dans ce cas n'appartiennent en propre à aucun des types d'entités qu'elle relie.

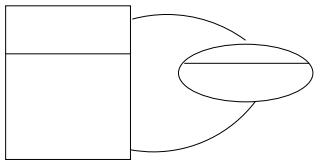



On distingue :

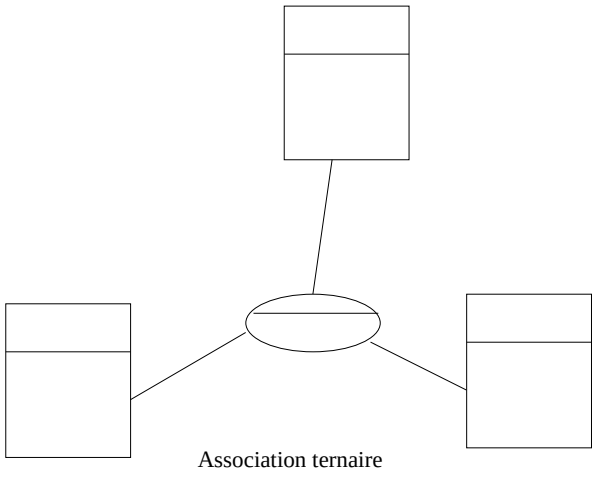
- Les associations binaires qui relient entre elles les différentes instances de deux types d'entité.
- Les associations n-aires (n > 2) qui relient les instances de n types d'entité.
- Les associations réflexives qui relient des instances d'un type d'entité avec d'autres instances du même type. On en trouve un exemple dans l'association Composant-Composé qui traduit le fait qu'une pièce de machine peut être un assemblage de plusieurs autres composants, et réciproquement.



Association binaire



Association réflexive

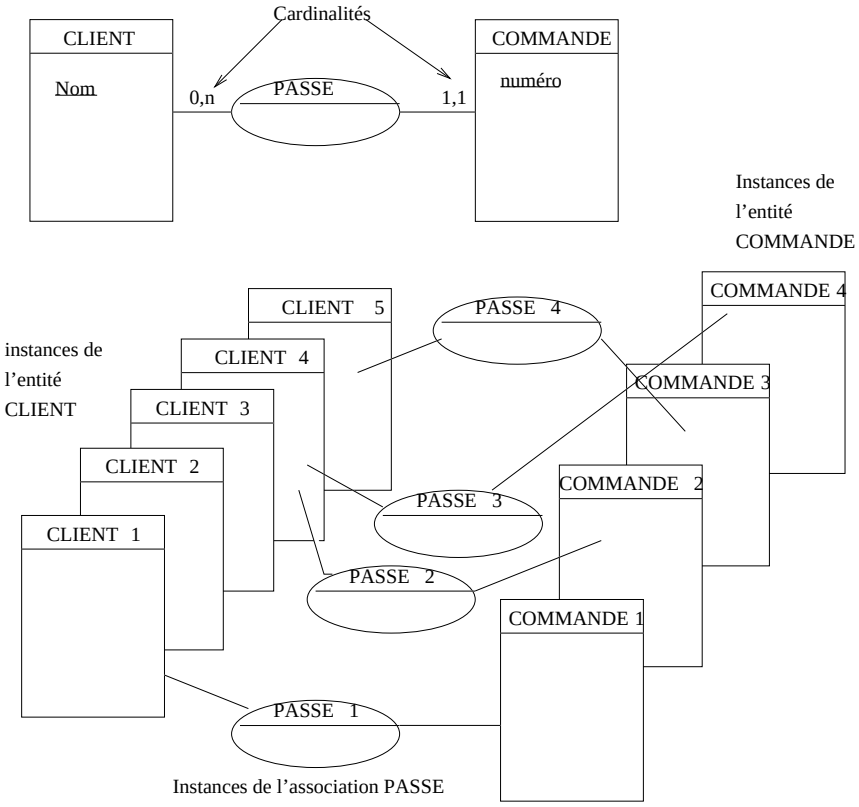


Association ternaire

### 1.2.1 Cardinalités

Les cardinalités d'une association sont un élément clé de la traduction du MCD vers un modèle logique. Elles permettent de préciser les nombres minimum, et surtout maximum d'occurrences d'une entité pouvant être impliquée dans les occurrences de l'association.

Il faut faire très attention au fait que les normes US et européennes sont inverses l'une de l'autre dans la représentation des cardinalités. En Europe, et en France plus particulièrement, les cardinalités sont placées vis à vis du type d'entité qu'elles renseignent. Ainsi, dans l'exemple ci-dessous, un client passe de 0 à plusieurs commandes, et une commande est passée par un client et un seul. L'interprétation de ces cardinalités seraient inversées si le schéma figurait dans un livre publié aux USA.




Les cardinalités doivent toujours être analysées du point de vue de l'entité à laquelle elles sont accolées. Les valeurs possibles sont : 0,1 0,n 1,1 et 1,n

Les cardinalités permettent de préciser la nature de l'association et du lien qu'elle constitue entre les entités. On parle de liens hiérarchiques, fonctionnels et maillés. Un lien fonctionnel est l'inverse d'un lien hiérarchique et relie des feuilles à la racine d'un arbre. Le lien maillé est le lien qui permet de relier entre-eux des nœuds au sein d'un graphe.

Les cardinalités minimales peremettent de préciser des éléments de l'annalyse, mais ne jouent aucun rôle sur la traduction du MCD en un schéma logique. Lorsque les cardinalités *maximales* valent 1 et N de part et d'autre d'une association, il y a présence d'un lien hiérarchique (du côté du N) et d'un lien fonctionnel (du côté du 1). Le lien hiérarchique exprime le point de vue du sommet d'un arbre, le lien fonctionnel le point de vue des feuilles. Dans l'exemple de la figure, un client passe plusieurs commandes. Il est placé à la racine d'un arbre dont les commandes sont en feuilles. Chaque commande "voit" le client qui l'a émise.

Le lien maillé correspond à des cardinalités maximales valant N de part et d'autre de l'association. L'association `publie` figurant à la page précédente est construite sur un lien maillé : un auteur peut publier plusieurs ouvrages, mais il se peut qu'un ouvrage soit publié par plusieurs auteurs. Le lien maillé peut être implémenté par la réunion de deux liens fonctionnels opposés l'un à l'autre.

# 2 Règles d'écriture du MCD

- Les propriétés doivent être mono valuées. On ne peut rencontrer une propriété sous la forme d'une liste de valeurs ; elle est toujours représentée par une valeur unique. Dans le cas contraire, elle doit être décomposée sous la forme d'au moins deux entités associées.
- Les différentes valeurs des propriétés d'une entité doivent dépendre entièrement de l'identifiant (clé) de cette entité.
- Dans le cadre d'une association, deux entités de types différents sont reliées par une seule instance de l'association.
- Si la propriété d'une entité dépend d'une propriété autre que la clé, c'est le signe que l'analyse a été mal faite et qu'il existe un type d'entité imbriqué dans le type initial. Les propriétés ne peuvent dépendre que de la clé, sauf s'il existe plusieurs clés candidates dans l'entité. Ainsi, si l'on est certain de ne jamais avoir d'homonymes, un employé peut indistinctement être repéré par un numéro ou par son nom. Ces deux propriétés sont des clés candidates, et une seule (probablement le numéro) sera choisie comme clé primaire. La clé primaire est une clé d'*implémentation*. Au niveau du modèle (sur le plan théorique), les autres propriétés ne dépendent que de cette clé, mais au niveau logique (sur le plan pratique), elles continuent toutes de dépendre à la fois du numéro et du nom.

# Anciens modèles de données

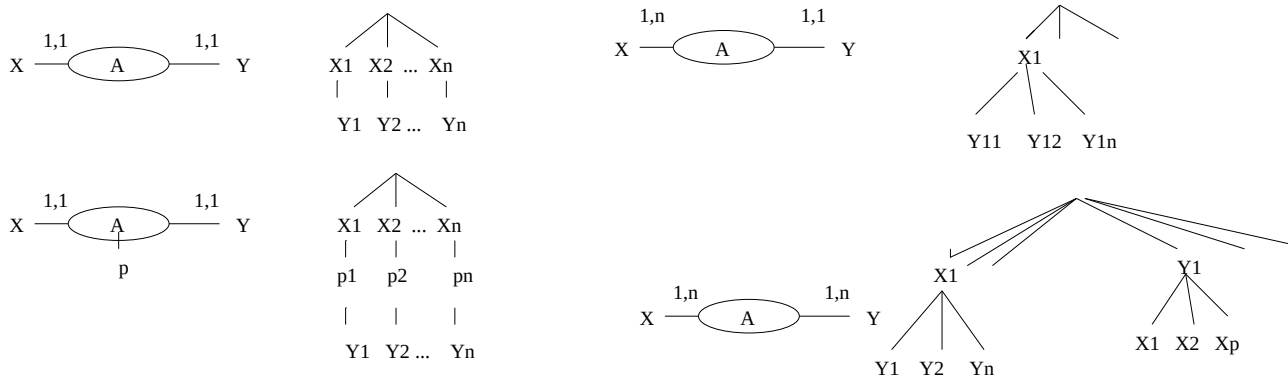
## 1 Le modèle hiérarchique

### 1.1 Généralités

Développé pour les besoins de la NASA, il a donné naissance entre autre à IMS (IBM) qui utilise, pour manipuler les données, un COBOL navigationnel.

Les entités du MCD sont représentées par des *segments de données*. Il n'y a pas de concept d'enregistrement. Les propriétés sont des champs, et le type de lien pris en compte est le lien hiérarchique. Le SGBD est une forêt composée d'arbres de segm ents et de pointeurs.

### 1.2 Traduction du MCD




Le mécanisme de traduction s'appuit sur l'arité et la cardinalité des associations. On distingue les cas suivants :

- Les associations binaires bi-univoques X $\leftrightarrow$ Y sont traduites par des arbres dont soit X soit Y est la racine, selon la sémantique des entités (fig. 1-a). Si l'association est porteuse de propriétés, celle-ci est représentée p ar un nœud constituant un niveau intermédiaire entre X et Y (fig. 1-b).
- Les associations binaires hiérarchiques X $\leftrightarrow$ Y sont traduites par des arbres dont X est obligatoirement la racine (fig. 1-c).
- les associations n-aires ou construites sur des liens maillés sont traduites par une série d'arbres (forêt). L'un d'eux est constitué des données (segments), les autres étant des arbres de pointeurs, fournissant un accès hiérarchisé à une informati on qui n'est pas dupliquée. Le SGBD est la racine (symbolique) qui permet de relier tous ces arbres en un arbre unique. (fig 1-d).

### 1.3 Anomalies

le modèle donne lieu à un certain nombre d'anomalies de fonctionnement.

- il est impossible d'insérer des données sans avoir au préalable créé la chaîne des nœuds supérieurs qui la relient à la racine.
- la suppression d'un nœud conduit nécéssairement à la destruction du sous arbre associé.
- la mise à jour conduit à balayer la totalité de la forêt dont le SGBD est la racine.

### 1.4 Avantages-inconvénients

- Le modèle conduit naturellement à une bonne représentation des systèmes hiérarchiques et fonctionne remarquablement avec des applications de cartographie et de CAO.
- il est simple et facile à implémenter.
- il est cependant mal adapté à la représentation des liens maillés et des réalités complexes.

– ses bonnes performances sont limitées par les anomalies de fonctionnement exposées ci-dessus.
– les bases hiérarchiques doivent être manipulées à l'aide de langages navigationnels lourds et coûteux à mettre en œuvre.
– l'indépendance entre les niveaux logiques et physiques est pratiquement inexistante.

# 2 Le modèle réseau

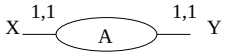
## 2.1 généralités

Ce modèle correspond à la deuxième génération de SGBD. Il a été normalisé par le DBTG CODASYL, et a donné lieu à de nombreux produits commerciaux (IDS (CII-HB), DBMS (DEC), IDMS). Les recommandations du CODASYL ont conduit à fixer la syntaxe et la sémanti que des opérations des langages de manipulation.

Au concept d'entité du MCD correspond la notion d'enregistrement. Les propriétés sont des *items* de données. Le type du lien pris en compte est le lien hiérarchique.

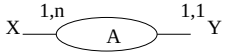
Le concept de base, sur lequel repose tout le modèle, est le concept de *COSET*, qui relie un type d'enregistrement *maître* avec un type d'enregistrement *membre*, et fait disparaître le besoin d'une clé pour accéder à un enregistrement parti culier.

## 2.2 traduction du MCD

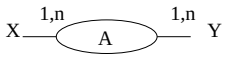



Coset XY : Maître : X (ou l'inverse)
membre : Y

Si A contient une propriété p, celle-ci est transférée au niveau de l'entité membre.



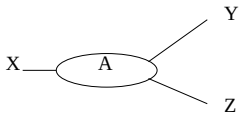

Coset XY : Maître : X
membre Y




Coset XA : Maître : X
membre : A

Coset YA : Maître Y
membre : A

A est une entité artificielle




Coset XA Coset YA Coset ZA A est une entité artificielle

Le mécanisme de traduction est schématisé ci-dessus. Il fait intervenir, pour la traduction des liens maillés ou des associations n-aires (avec n > 2), une entité "bidon", éventuellement vide, qui n'est là que pour permettre la décomposition du lien ma illé en une série de liens hiérarchiques.

## 2.3 propriétés du COSET

– une occurence donnée d'un COSET possède un propriétaire et plusieurs membres. Les types d'entités du propriétaire et des membres sont nécessairement différents.
– un type d'enregistrement peut être propriétaire dans plusieurs instances de COSETS différents, mais pas dans des instances différentes du même COSET.
– un type d'enregistrement peut être membre dans plusieurs instances de COSETS différents, mais pas dans des instances différentes du même COSET.

## 2.4 Avantages - inconvénients

– Le modèle assure une bonne représentation des liens maillés.

- Il permet de ce fait une meilleure représentation des réalités plus complexes que les organisations hiérarchiques.
- il est dépourvu d'anomalies de fonctionnement.
- Il a donné lieu à une offre commerciale assez étoffée.
- Les moteurs de bases de données réseaux ont été largement utilisés dans la conception de produits hyper-textes, en association avec des index arborescents.
- il conduit malheureusement à une indépendance faible entre les niveaux logiques et physiques.
- Les données ne peuvent être manipulées qu'avec des outils de navigation lourds et coûteux.

## 2.5 Langage navigationnel

Le système dispose d'une multitude de pointeur :
- un par type d'enregistrement.
- un par type de COSET.
- un pour l'enregistrement courant.

les échanges de données se font à l'aide de *gabarits d'enregistrements*.

De nombreux éléments syntaxiques des langages permettent de positionner ces pointeurs sur telle ou telle partie de l'information.

- recherche par nom ou critère : La recherche du premier enregistrement de nom donné se fait par :
  `FIND enregistrement RECORD USING valeur;`
  La recherche des autres enregistrements de même nom utilise :
  `FIND NEXT DUPLICATE enregistrement RECORD [USING condition]`
  En cas d'erreur, une condition (`STATUS CHECK`) est levée et permet d'interrompre le traitement.
- Recherche d'un membre d'un COSET :
  `FIND PRIOR | NEXT | FIRST | LAST | n enregistrement RECORD  WITHIN coset SET`
- Recherche du propriétaire : `FIND OWNER RECORD OF coset SET`
- Recherche au sein d'un COSET :
  `FIND enregistrement RECORD VIA coset SET [WHERE condition]`

Le LMD dispose aussi d'instructions de boucles de la forme :

`loop until condition ... end loop`

La condition de boucle pouvant être : `no more records` ou encore `end of set`

La programmation consiste à naviguer à travers le graphe de la base et conduit à des programmes rapidement assez complexes, qui rendent difficiles la réalisation d'applications à la carte.

Exemple d'utilisation pour les type d'entités définis ci-dessous avec leurs propriétés :

```
Stations(ns,nom,nr)
Region(nr,nom)
Activités(na,libelle)
propose(ns,na)
```

Les Cosets correspondants sont :

```
AP : Maitre = Activites, membre = propose
RS : Maitre = Region, membre = Station
SP : Maitre = Station, membre = Propose
```

Procédure : *Activités des stations des alpes d'altitude > 1200m*

```
find region record using nom='ALPES';
exit if status_check;
find station record via RS set where altitude > 1200;
exit if status_check;
loop until no more records
  find first propose record within SP set;
  exit if status_check;
  find owner record of AP set;
  output libelle of activite;
  loop until end of set
    find next propose record within SP set;
    exit if status_check;
    find owner record of AP set;
    output nom of activite;
  end loop;
  find next duplicate station record using altitude > 1200;
end loop;
```

# Le modèle relationnel

## 1 Généralités

### 1.1 Historique

Il est né de la thèse de Codd (IBM San José - 1970) et il est basé sur des concepts très simples. De 72 à 75, les laboratoires IBM ont conçu un premier prototype : SYSTEM R, d'où sont directement issus les logiciels : INGRES (RTI - 1976), ORACLE (1979), SQL/DS (IBM-1981) et DB2 (IBM - 1983). Pour manipuler System R, des prototypes de langages ont été créés qui ont rapidement abouti à SQL. Normalisé par l'ANSI en 86, ce langage a continué son développement pour donner lieu en 92 à la norme SQL-2. Une nouvelle et dernière normalisation a eu lieu en 1999 (SQL-3).

### 1.2 Définitions

Au type d'entité du MCD correspond la notion de *Relation*, à ne pas confondre avec une association. Une relation, dans le cadre du modèle, est assimilée à une table. Les propriétés sont appelées des *attributs*. Le seul type de lien reconnu par le modèle est le *lien fonctionnel* entre N instances d'un type d'entité et une et une seule instance du type d'entité associé.

Un *Domaine* est un ensemble de valeurs : ensemble des entiers, ensemble des couleurs (bleu, blanc, rouge), intervalle 0..1, etc ...

Le produit cartésien de plusieurs domaines D1, D2, ..., Dn est constitué d'un ensemble de n-uplets encore appelés *tuples* : (v1, v2,..., vn), où chaque vi est une valeur d'un Di.

| **D1** | **D2** |
|---|---|
| Bleu | 0 |
| Bleu | 1 |
| Blanc | 0 |
| Blanc | 1 |
| Rouge | 0 |
| Rouge | 1 |

*Produit cartésien de l'ensemble D1 des couleurs par celui D2 des valeurs binaires*

Une relation est un sous ensemble du produit cartésien des domaines sur lesquels sont définis ses attributs. Elle peut donc être représentée par une table à 2 dimensions dont les colonnes correspondent aux différents domaines et dont les lignes représentent les tuples.

Le *schéma relationnel* est constitué de l'ensemble des relations définies par leur nom suivi de la liste de leurs attributs. Exemple de schéma relationnel :

`EMPLOYE(numero, nom, adresse, salaire, emploi, departement)`

Il n'y a pas d'identificateur de tuple. C'est un attribut particulier, la clé primaire (numero dans l'exemple précédent), qui permet d'identifier une rangée de la table parmi toutes les autres. Il arrive que plusieurs attributs soient capables de jouer le rôle de clé pour une relation. L'un d'eux est alors choisi pour constituer la *clé primaire*, les autres *clés candidates* deviennent alors des *clés secondaires*.

Le schéma de chaque relation représente son *intention*, alors que les données du tableau constituent une *extension* particulière, c.à.d un instantané des valeurs qu'il contient à un instant donné. La figure précédente représente une extention du schéma (couleur, nombre binaire), alors que le schéma relationnel de la table `employe` représente l'intention de cette même table.

### 1.3 Traduction

La traduction du MCD en un schéma relationnel est un processus extrêmement simple :

– Chaque relation est traduite par une table.

– Les associations construites sur des liens maillés sont représentées par des tables, et elles seulement. Leur clé primaire est au minimum obtenue par la réunion des clés des relations associées. Il peut arriver que ce regroupement ne soit pas suffisant pour garantir l'unicité des valeurs sur chaque ligne et qu'il faille y rajouter des propriétés supplémentaires de l'association.
– Les associations construites sur des liens fonctionnels sont implémentées de la manière suivante : Dans la table qui représente le type d'entité situé *du côté du lien fonctionnel*, on ajoute un attribut représentant un *pointeur par valeur* vers la clé de la table qui implémente l'autre type d'entité. Dans l'exemple Clients-Commandes du chapitre sur le MCD, on place dans la table `commandes` un attribut permettant de faire référence à la clé du client ayant passé la commande. Si l'association est porteuse de propriétés, celles-ci suivent la représentation de l'association. Une propriété éventuelle de l'association `passe` serait représentée par l'ajout de l'attribut correspondant dans la table `commandes`.

## 1.4 Règles d'intégrité

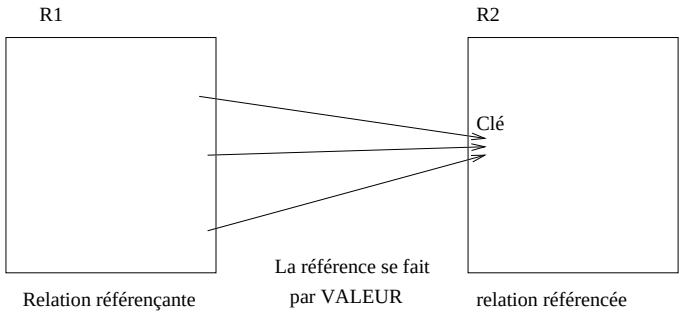
Ces règles garantissent le bon fonctionnement du modèle et l'accès aux données élémentaires représentées par les lignes des tables qui traduisent les relations.

### 1.4.1 L'intégrité d'unicité

L'accès à une ligne particulière d'une table ne peut être réalisé directement que par l'intermédiaire de la valeur d'une clé qui joue le rôle d'un identifiant. Le modèle relationnel est un "modèle valeur" parce que ce sont des valeurs qui servent à l'identification ; le corollaire, c'est qu'une telle valeur doit obligatoirement être unique et définie. Lorsqu'aucune valeur ne peut convenir pour repérer la rangée d'une table, il faudra en créer une, même artificiellement, en lui attribuant par exemple un numéro qui n'aura qu'une signification interne au sein du système.

### 1.4.2 L'intégrité de référence

Cette contrainte traduit l'existence d'un lien fonctionnel entre deux relations. La relation qui est du côté de la cardinalité *1,1* dans l'association est la relation référençante, celle qui est racine de la hiérarchie (cardinalité *1,n*) est la relation référencée. On doit trouver dans la relation référençante un attribut construit sur le même domaine que la clé de la relation référencée. Une ou plusieurs valeurs de cet attribut doivent correspondre à une valeur unique de la clé. Ce procédé permet d'associer par valeur plusieurs rangées de la relation référençante à une et une seule ligne de la relation référencée.




### 1.4.3 L'intégrité de domaine

Les valeurs d'un attribut doivent obligatoirement correspondre aux valeurs du domaine sur lequel il est défini. Un domaine est souvent associé à des critères logiques qui permettent de restreindre le type sur lequel il est construit (entiers positifs, caractères limités aux voyelles, etc). Le respect de l'intégrité de domaine permet donc de contrôler lors de l'insertion ou de la mise à jour d'une valeur que celle-ci est conforme au cahier des charges (un salaire ne peut être négatif, un mois doit être compris entre 1 et 12, un nom sera transformé indépendemment de sa saisie avec une initiale en capitale, etc ...)

Un schéma totalement relationnel est un schéma satisfaisant tous ces types d'intégrité.

### 1.5 Avantages et inconvénients du modèle

- Le modèle relationnel est un modèle simple pour l'utilisateur qui ne manipule que des tables.
- L'indépendance entre les niveaux peut être parfaitement respectée.
- Le représentation est uniforme. Elle possède une certaine puissance que lui confert la théorie algébrique qui la fonde.
- Le concept de table permet de garantir des accès sécurisés aux données, en ligne comme en colonne.
- Il existe de nombreux interfaces non procéduraux, un grand nombre de langages conviviaux de manipulation et un standard (SQL).
- L'offre commerciale est énorme et concerne tous les systèmes, toutes les plates-formes.
- Néanmoins, le processus de *normalisation* pose certaines difficultés (anomalies de fonctionnement, baisses de performances) et fait perdre un peu de l'indépendance entre le niveau conceptuel et le niveau logique.
- Le modèle est inadapté au traitement de données multimédia.

## 2 L'algèbre relationnelle

Elaborée par Codd dans sa thèse, elle repose sur un petit nombre d'opérateurs de type ensemblistes. Les exemples qui vont suivrent manipulent les relations suivantes, R, S et T respectivement :

**R**

| A | B | C |
|---|---|---|
| a | 1 | a |
| b | 1 | b |
| a | 1 | d |
| b | 2 | f |

**S**

| A | B | C |
|---|---|---|
| a | 3 | f |

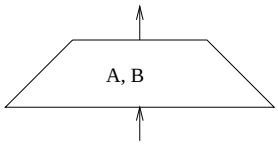
**T**

| B | C | D |
|---|---|---|
| 1 | a | 1 |
| 3 | b | 1 |
| 3 | c | 2 |
| 1 | d | 4 |
| 2 | a | 3 |

### 2.1 Opérateurs monadiques

Ces opérateurs sont des réducteurs de données.

#### 2.1.1 Projection



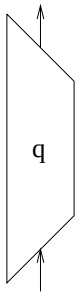

*Symbole graphique de l'opérateur de projection*

La projection est un opérateur d'accès par les colonnes. La projection U d'une relation R sur une liste d'attributs est la relation dont le schéma se réduit aux attributs de la liste. Ses tuples sont ceux de R, avec élimination des tuples en double.

| A | B |
|---|---|
| a | 1 |
| b | 1 |
| b | 2 |

$$\mathrm{U} = \prod_{A,B} R$$

### 2.1.2 Restriction




*Symbole graphique de l'opérateur de restriction*

La restiction est un opérateur d'accès par les lignes. Une *formule de restriction* ou *qualification* q est une expression logique reliant des attributs de la relation opérande avec des constantes par l'intermédiaire d'opérateurs de comparaison ; par exemple :

$A = a$ AND $B < 2$

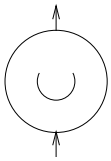
La restriction U de la relation R par la qualification q est une relation de même schéma que R dont les tuples sont ceux de R qui vérifient la qualification.

| **A** | **B** | **C** |
|---|---|---|
| a | 1 | a |
| a | 1 | d |

$\mathrm{U} = \sigma_{A=a}\ (R)$

## 2.2 Opérateurs dyadiques

### 2.2.1 Union



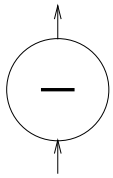
*Symbole graphique de l'opérateur d'union*

L'union de deux relations R et S *de même schéma* est la relation de même schéma dont les tuples sont à la fois ceux de R et de S. L'union est un opérateur expanseur de données.

| **A** | **B** | **C** |
|---|---|---|
| a | 1 | a |
| b | 1 | b |
| a | 1 | d |
| b | 2 | f |
| a | 3 | f |

U= R $\bigcup S$

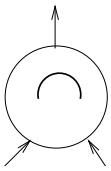
### 2.2.2 Différence



*Symbole graphique de l'opérateur de différence*

La différence de deux relations de même schéma R et S est une relation de même schéma dont les tuples sont ceux de R n'appartenant pas à S. La différence n'est pas un opérateur commutatif.

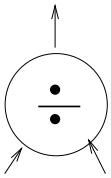
### 2.2.3 Intersection



*Symbole graphique de l'opérateur d'intersection*

L'intersection de deux relations R et S de meme schéma est une relation de même schéma constituée des tuples qui appartiennent à la fois R et à S. L'intersection, obtenue par composition de différences, n'est pas un opérateur de base.

### 2.2.4 Division



*Symbole graphique de l'opérateur de division*

La division d'une relation R par une relation S est la relation formée des tuples qui, concaténés à chaque tuplede S, redonnent un tuple de R. La division n'est pas un opérateur de base et peut être réalisée par l'expression algébrique :

$$R \div S = \prod_{X}(R) - \prod_{X}((\prod_{X}(R) \times S) - R)$$

où X est l'ensemble des attributs de R qui ne sont pas dans S.

Soit la relation Z :

| **B** | **C** |
|---|---|
| 1 | a |
| 1 | d |

La division R ÷ Z est :

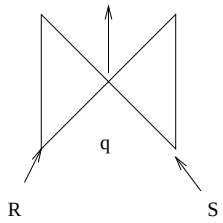
| **A** |
|---|
| a |

### 2.2.5 Produit cartésien

Le produit cartésien de deux relations R et S est une relation dont le schéma est la concaténation de ceux des relations composantes et dont les tuples sont obtenus en combinant chaque tuple de R avec tous ceux de S.

Le produit cartésien (ainsi que tous les opérateurs dérivés) est un expanseur de données.

### 2.2.6 Jointures




*Symbole graphique de l'opérateur de jointure*

On distingue plusieurs opérateurs de jointures. Ils sont associés à des formules de qualification reliant entre eux certains attributs des relations composantes. On distingue :

- l'équijointure : La clause de qualification est une égalité entre deux attributs des relations opérandes. La relation résultante est le sous-ensemble du produit cartésien réduit aux tuples vérifiant la qualification, sans répétition, dans le schéma, de l'attribut commun.

$$U = R \underset{R.A = T.C}{\bowtie} T$$

| A | R.B | R.C | T.B | T.D |
|---|---|---|---|---|
| a | 1 | a | 1 | 1 |
| a | 1 | a | 2 | 3 |
| b | 1 | b | 3 | 1 |
| a | 1 | d | 1 | 1 |
| a | 1 | d | 2 | 3 |
| b | 2 | f | 3 | 1 |

- la Θ-jointure : dans la clause de qualification, l'opérateur d'égalité est remplacé par un des cinq autres opérateurs de comparaison.

$$U = R \underset{R.B < T.D}{\bowtie} S$$

- l'autojointure : c'est la jointure d'une relation avec elle-même. SI algébriquement parlant, il est possible d'écrire :

$$U = R \underset{B < B}{\bowtie} R$$

  dans la pratique, il conviendra de donner des *noms ou alias* différents aux deux opérandes, puisque les traitements vont concerner des lignes différentes.
- la jointure naturelle : C'est l'équijointure de deux relations sur tous les attributs ayant même nom et construits sur le même domaine.
- les semi-jointures : on distingue la semi–équijointure, la semi-Θ jointure, la semi-jointure naturelle. Une semi jointure entre R et S : $R \ltimes_{A = A} S$, est la projection sur l'opérande gauche de la jointure correspondante. On a la relation :

$$R \ltimes_{A = A} S = \prod_{Attributs\ de\ R} (R \underset{R.A = T.A}{\bowtie} S)$$

  La semi-jointure de deux relations R et S est la relation de même schéma que R dont les tuples sont ceux de R qui participent à la jointure de R et de S.
  La semi-jointure est un opérateur réducteur de données et n'est pas commutative.

## 2.3 Composition d'opérations

La composition des divers opérateurs permet d'envisager des traitements algébriques complexes autorisant des manipulations formelles des données.

Soient les relations :

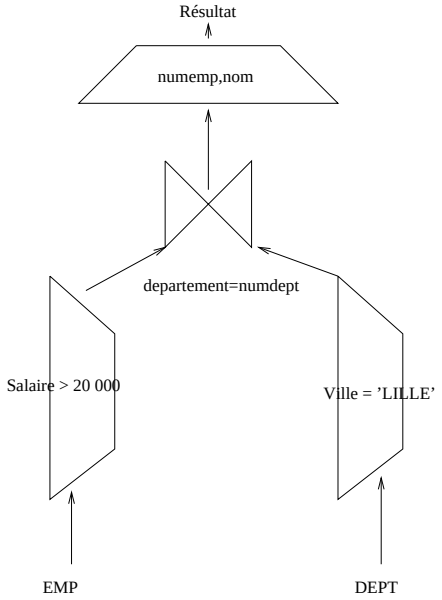
`EMP(numemp, nomemp, salaire, emploi, departement)`

`DEPT(numdept, nomdept, adresse, ville)`

La détermination des informaticiens basés à Lille et gagnant plus de 20000 F se fera par :

$$\prod_{numemp,nom} (\sigma_{salaire>20000}\ (EMP) \underset{departement=numdept}{\bowtie} \sigma_{ville='LILLE'}\ (DEPT))$$

Cette expresion peut être représentée graphiquement par l'arbre statique d'exécution suivant :




# 3 Normalisation

## 3.1 Quel est le bon schéma ?

Il n'y a pas une solution unique à un problème de gestion de données. En général, des équipes d'analystes indépendantes permettraient d'aboutir à des dizaines de MCD différents pour un même problème complexe. Une fois que l'on applique les règles automatiques de traduction présentées au début de ce chapitre, on obtient des schémas relationnels aux propriétés bien différentes. Au début du développement des bases de données relationnelles, ces schémas pouvaient aboutir à des requêtes d'interrogation qui s'exécutaient en des temps variant de moins d'une seconde à plusieurs dizaines de minutes. Très tôt s'est donc posé un problème d'évaluation des schémas et de classification de leurs propriétées.

C'est par l'introduction du concept de dépendance fonctionnelle qu'on a pu construire des outils d'évaluation et de manipulation des schémas relationnels. On a pu ainsi répondre aux questions : "qu'est-ce qu'un bon schéma", "quel est le meilleur schéma", "comment améliorer un schéma ".

## 3.2 Dépendance fonctionnelle

### 3.2.1 généralités

Les dépendances fonctionnelles (D.F.) permettre d'introduire des dépendances de *valeurs* entre les données. C'est une notion intuitive : un salaire dépend (en général) de la qualification d'un employé, la vitesse maximale d'un véhicule dépend de sa puissance. Si la date de naissance dépend d'un individu, elle dépend donc de la clé (éventuellement un numéro) qui, dans la base, sert à le repérer.

On aura dans ce cas : numero → date_naissande.

Dans toute relation R(X,Y) munie de la DF X → Y, X est *clé candidate*. Si elle est la seule possible, ou si elle choisie parmi d'autres possibles, elle devient la *clé primaire* de la relation R.

On appelle *dépendance élémentaire* X → Y, toute dépendance telle que Y ne dépend pas d'un sous-ensemble de X.

C'est un modèle de dépendance de valeur. La valeur d'une donnée dépend de celle d'une autre donnée. (**Attention** : *l'inverse n'est pas forcément vrai*! La détection d'une conséquence n'implique pas qu'elle

soit due à une cause spécifique, sauf lorsque le mécanisme causal, auquel s'apparente le mécanisme des DF ne dépend que d'une cause unique). Ceci veut dire que si on fait dépendre un âge d'un nom, le même nom devra toujours être associé à la même valeur de l'âge. La dépendance impose ainsi au nom d'être associé au même âge qui doit rester fixe, et tous les homonymes devront avoir cet âge. Il convient donc, lors de la modélisation, d'examiner cette notion de dépendance en faisant preuve de beaucoup de sens critique.

### 3.2.2 Axiomes d'Armstrong

Le formalisme des dépendances fonctionnelles est celui d'Armstrong (1974).

X,Y,Z désignant des ensembles d'attributs, on dit que Y dépend fonctionnellement de X, et on note $X \rightarrow Y$, s'il existe une DF de X vers Y.

1. Réflexivité : si $Y \subseteq X$ alors $X \rightarrow Y$
2. Augmentation : Si $X \rightarrow Y$ et si W est un ensemble quelconque d'attributs, alors $XW \rightarrow YW$
3. Transitivité : Si $X \rightarrow Y$ et $Y \rightarrow Z$, alors $X \rightarrow Z$
4. Pseudo-transitivité : si $X \rightarrow Y$ et $YW \rightarrow Z$, alors $XW \rightarrow Z$
5. Union : si $X \rightarrow Y$ et $X \rightarrow Z$, alors $X \rightarrow YZ$
6. décomposition : Si $X \rightarrow YZ$, alors $X \rightarrow Y$ et $X \rightarrow Z$

l'axiome d'augmentation permet d'intégrer des attributs isolés, non liés à une dépendance fonctionnelle particulière. Les trois dernières règles ne sont pas indispensables ; ce sont des règles opératoires permettant, plus directement, la manipulation des graphes de dépendances.

Ces différentes DF génèrent un graphe F. On appelle *fermeture transitive* d'un graphe le graphe obtenu en ajoutant à F l'ensemble de tous les arcs qui se déduisent de ceux de F par les 3 premiers axiomes. Ce graphe $F^+$ définit la *couverture maximale* des DF.

Soit R(A,B,C,D) une relation munie des DF suivantes :
A → BC
C → D
D → B
la fermeture transitive sera obtenue par l'ajout de :
A → D
au graphe initial.

La *couverture minimale* est obtenue par l'ensemble des arcs de $F^+$ qui ne sont pas redondants. C'est ce graphe qui permet d'obtenir une décomposition de F sans perte d'information (cf. plus loin).

Deux graphes F et G obtenus à partir de la même liste de DF correspondent à une solution correcte (mais dont les propriétés ne sont pas équivalentes) si $F^+ = G^+$

En pratique, le graphe des dépendances fonctionnelles permet d'obtenir, de façon algorithmique, le schéma conceptuel ( ou une version de ce schéma), et la réciproque est vraie aussi.

## 3.3 Décomposition

Puisqu'on peut toujours transformer un graphe de DF en sa fermeture transitive, on peut imaginer de placer tous les attributs du problème dans une seule et même relation universelle U. On constate que la gestion d'une base relationnelle organisée selon ce principe conduit à une série de disfonctionnements qui la rendent rapidement incohérente et inefficace. On s'aperçoit qu'une décomposition de U en plusieurs relations regroupant les DF autours de leur source améliore le fonctionnement de la base. Cette décomposition, appliquant l'axiome 6, doit être réversible (axiome 5) et sans perte d'information. Elle utilise pour ce faire les opérations de projection et de jointure qui réalisent respectivement ces deux opérations de décomposition et d'union.

On dit qu'une décomposition (par projection) est *sans perte d'information* s'il existe des jointures qui permettent de reconstituer les données sous leur forme initiale. En d'autres termes, si on décompose U en :

$U_1 = \prod_X U$, $U_2 = \prod_Y U$ et $U_3 = \prod_Z U$, on doit obligatoirement avoir :

$$U = U1 \bowtie U2 \bowtie U3$$

Soit par exemple :

| A | B | C | D |
|---|---|---|---|
| a | 5 | x | 2 |
| B | 5 | y | 2 |
| C | 5 | x | 2 |

On voit bien que, dans cette relation R, les DF énoncées au paragraphe précédent sont vérifiées.
La décomposition `R1(A,B)` et `R2(B, C, D)` n'est pas conservative puisque $R1 \bowtie R2 \neq$ R.
Par contre `R1(A, B, C)` et `R2(A,D)` constitue bien une décomposition sans perte.

## 3.4 Formes normales

Ce processus de décomposition permet de mettre en évidence des situations caractérisées par la présence ou non d'anomalies de fonctionnement. Ces situations fournissent des critères d'évaluation du comportement d'une base relationnelle.

### 3.4.1 Première forme normale

Une relation est en première forme normale si tous ses attributs contiennent des valeurs atomiques. Sont exclus les attributs construits sous formes complexes (vecteurs, listes, arbres, objets ...)

Dans la relation `(numemp, nom, enfant(prenom,dateNaissance))`

l'attribut `enfant` est un groupe composé qui ne peut être pris en compte dans le modèle relationnel. Une telle "relation" appartient au monde orienté-objet (OO) et ne peut être manipulée par les opérateurs relationnels. Le modèle relationnel est un modèle ensembliste, et tous les éléments d'un ensemble doivent avoir la même cardinalité, faute de quoi on ne peut plus parler d'ensemble au sens mathématique du terme.

Il est nécessaire de décomposer la relation précédante en deux relations qui seront alors en première forme normale :

`Emp(numemp, nom)` et `Enfants(numemp, prenom, dateNaissance)`

### 3.4.2 Deuxième forme normale

Cette forme normale ne s'applique qu'aux relations possedant une clé multiple.

Une relation est 2NF ssi :

- elle est 1NF
- tous ses attributs non-clé dépendent de la totalité de la clé.

La relation `Stock(piece, entrepot, qté, adresse)` n'est pas 2NF car l'adresse ne dépend que de l'entrepôt. Elle subit de ce fait des anomalies de fonctionnement qui sont :

- une redondance d'informations susceptible d'induire des incohérences ;
- en cas de mise à jour de l'adresse d'un entrepôt, l'obligation de balayer toute la table pour modifier cette dernière partout où elle peut être présente ;
- lors de la suppression de la dernière pièce de l'entrepôt, on perd toute trace de celui-ci.

Si on désire mettre la relation en 2NF, on doit la décomposer en :

`Stock(piece, entrepot, qté)`
`Local(entrepot, adresse)`

### 3.4.3 Troisième forme normale

Une relation est 3NF ssi :

- elle est 2NF
- Aucun attribut non-clé ne dépend fonctionnellement d'un autre attribut non-clé.

Soit : `Emp(numemp, nom, service, batiment)`

Le service implique le bâtiment dans lequel il se situe. De ce fait, Emp n'est pas 3NF. Elle dissimule la présence d'une relation cachée qui a échappé à l'analyse, et doit être décomposée en :

`Emp(numemp, nom, service)`
`Localisation(service, batiment)`

La décomposition est bien 3NF. En pratique, cette forme normale est importante car dotée de propriétés correctes. Il existe souvent plusieurs façons d'opérer des décompositions 3NF, mais toute relation

en possède au moins une qui soit sans perte et qui préserve les DF. Un algorithme dû à Bernstein permet de la générer automatiquement.

Le principe de la normalisation des relations s'appuie sur la notion de clé. On distingue la clé primaire parmi toutes les clés candidates possibles (les clés candidates non retenues sont appelées clés secondaires). La normalisation en 3NF prend en compte la notion générale de clé, qu'elle soit primaire ou secondaire. La seule différence qui intervient entre différentes clés correspond à un choix d'implémentation, mais toute clé candidate joue le même rôle que la clé primaire vis à vis des autres attributs de la relation.

### 3.4.4 Forme normale de Boyce-Codd

Dans la définition des formes normales qui précèdent, on ne s'est pas préoccupé des dépendances transitives (cachées) qui peuvent causer des anomalies de mise à jour. Celles-ci peuvent se manifester de façon insidieuse par l'intermédiaire de clés candidates qui pourraient demeurer dans une relation. La forme normale de Boyce Codd doit donc prendre en compte toute clé (au sens large) dans la relation.

Une relation est BCNF *si et seulement si* les seules DF élémentaires sont celles dans lesquelles il n'y a pas d'autre déterminant que la clé primaire. Une relation BCNF est aussi 3NF et constitue le degré ultime de décomposition sur la base des dépendances fonctionnelles.

En général, une décomposition BCNF aboutit à une perte de dépendance. Soit la relation :
`Entretiens(candidat, datexam, heure, jury, salle)`
avec les dépendances :
*candidat, datexam* → *heure, jury, salle*
*jury, datexam, heure* → *candidat*
*jury, datexam* → *salle*
La relation ci-dessus contient des dépendances transitives partielles et nécessite, pour éviter des anomalies de mise à jour, la transformation en deux relations BCNF :
`Entretiens(candidat, datexam,heure,jury)`
`Planning(jury, datexam, salle)`
La seconde DF est perdue.

### 3.4.5 Dépendances multivaluées

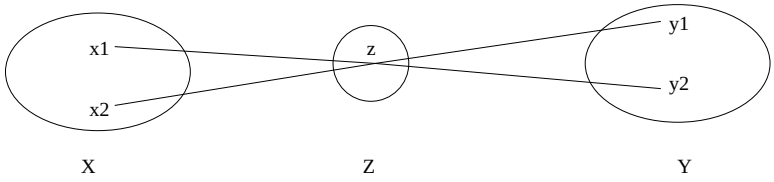
La décomposition BCNF est insuffisante pour éliminer totalement toutes les anomalies de fonctionnement et les redondances.

La relation `Etudiants(numetud, cours, sport)` indique quels cours suit et quels sports pratique un étudiant. Elle n'est pas 3NF et il n'y a même pas de dépendance fonctionnelles entre les données, ainsi qu'on peut le constater sur l'extension suivante où aucune clé ne peut être trouvée :

| **Numetud** | **Cours** | **Sport** |
|---|---|---|
| 100 | BD | Tennis |
| 100 | BD | Football |
| 200 | BD | Vélo |
| 200 | CL | Footing |

En fait, on trouve des dépendances multivaluées, les valeurs des attributs `cours` et `sport` étant intercorrellées par l'attribut `numetud`.

On dit qu'il y a *multi-détermination* entre des ensembles d'attributs X et Y ou encore que X multi-détermine Y ($X \twoheadrightarrow Y$) lorsqu'il existe un ensemble Z tel que, si $z \in Z$, $x1 \in X$, $x2 \in X$, $y1 \in Y$, et $y2 \in Y$, si $x_1zy_1$ et $x_2zy_2$ sont présents dans la relation R, alors $x_1zy_2$ et $x_2zy_1$ y sont aussi représentés (ou susceptibles de l'être).




Ces dépendances multivaluées (DM) sont régies par les axiomes suivants :
– Complémentation : $X \twoheadrightarrow Y \Rightarrow X \twoheadrightarrow R - X - Y$
– Multi-augmantation : Si $X \twoheadrightarrow Y$, et si W est un ensemble d'attributs de R, alors $XW \twoheadrightarrow YW$

- Pseudo-transitivité : si $X \twoheadrightarrow Y$ et $Y \twoheadrightarrow Z \Rightarrow X \twoheadrightarrow Z - Y$
- Réplication : $X \rightarrow Y \Rightarrow X \twoheadrightarrow Y$
- Coalescence : $X \twoheadrightarrow Y$ et $Z \subseteq Y$, avec $W \subseteq R$ et $W \bigcap Y = \emptyset$ et $W \rightarrow Z$, alors $X \rightarrow Z$
- Union : si $X \twoheadrightarrow Y$ et $Y \twoheadrightarrow Z$, alors $X \twoheadrightarrow YZ$
- Intersection : si $X \twoheadrightarrow Y$ et $X \twoheadrightarrow Z$, alors $X \twoheadrightarrow Y \bigcap Z$
- Différence : si $X \twoheadrightarrow Y$ et $X \twoheadrightarrow Z$, alors $X \twoheadrightarrow Y - Z$ et $X \twoheadrightarrow Z - Y$

Cet ensemble de règles est fermé et complet. On en déduit les règles de décomposition.

### 3.4.6 Quatrième forme normale

Un ensemble d'attributs muni de dépendances multivaluées est décomposable sans perte d'information. Cette décomposition est 4NF SSI toutes les dépendances multivaluées sont celles dans lesquelles une clé détermine un attribut. Avec l'exemple précédant, on obtient une décomposition en deux relations 4NF :

`Etudes(numetud, cours)`
`Activites(numetud, sport)`

### 3.4.7 Dépendance de jointure

La décomposition 4NF n'est pas suffisante pour éliminer les anomalies, car on peut encore trouver des redondances d'information, sans pour autant les devoir à des dépendances multivaluées. Ainsi, dans la relation `enseignement` suivante, la présence des tuples (X COO Z) et (X SYSTEME T) n'implique aucunement qu'on puisse trouver (X SYSTEME Z) ou (Y COO T).

**Enseignement**

| ETUDIANTS | COURS | ENSEIGNANTS |
|---|---|---|
| X | COO | Z |
| X | COO | T |
| X | SYSTEME | T |
| Y | SYSTEME | T |

En effet, les enseignants sont indépendants des étudiants, et ne peuvent inter-corréler l'association d'un étudiant et des cours qu'il suit.

On ne peut décomposer la relation initiale en deux, il faut nécessairement 3 relations de schémas :
`R1 = (Etudiant, Cours)`, `R2 = (Etudiant, Enseignant)`, `R3 = (Cours, Enseignant)`
Dans ce cas, `Enseignement = R1` $\bowtie$ `R2` $\bowtie$ `R3`

C'est ce qu'on appelle la dépendance de jointure (notée *), dont les dépendances multivaluées sont un cas particulier, et qui exprime une interdépendance de plusieurs entre eux.

### 3.4.8 Cinquième forme normale

Une relation R est 5NF si toute dépendance de jointure est impliquée par ses clés candidates.

Soit R(X,Y,Z,T), X et Y étant des clés candidates.

Une décomposition 5NF sans perte est alors :

*[(XY), (XZ), (XT)] ou encore *[(XY), (YZ), (YT)]

Une relation 5NF ne donne lieu à aucune anomalie et cette forme constitue l'ultime stade de décomposition d'une relation.

# Les langages du relationnel

La manipulation des données par un langage approprié a tout d'abord été proposée sous une forme algébrique dans la thèse de Codd et a constitué une première approche ensembliste. Divers prototypes de languages ont été élaborés au sein des laboratoires IBM pour assurer une interface plus conviviale aux utilisateurs : `Square`, puis `Sequel` et enfin `SQL` (1976) pour `SystemR`.

Une autre approche, prédicative, a été menée autour des algèbres de tuple et de domaine. Sur la base d'un premier langage, `Alpha`, on a vu apparaître successivement `Quel` et surtout `QBE`, qui a été par la suite utilisé par `Paradox`(Borland) et `Access`(MicroSoft).

## 1 Approche ensembliste

Cette approche est avant tout illustrée par le langage SQL, utilisé par la société Oracle dès 1979, puis par IBM dans `SQL/DS`(81) et `DB2`(83).

Ce langage a été normalisé par l'ANSI en 86. Une version améliorée, prenant en compte un certain nombre d'avancées proposées par des éditeurs, a été normalisée en 92. Elle inclut entre autre la prise en compte des contraintes relationnelles lors de la création des objets, le concept de curseur et celui de requêtes dynamiques. La norme 3 date de 1999. Elle a introduit l'emploi de procédures stockées(boucles et récursivité), les triggers et les principaux éléments du modèle Relationnel-Objet.

Les principaux éléments du langage sont exposés dans le poly de TP : *guide de l'utilisateur*.

## 2 Approche prédicative

### 2.1 Calcul relationnel de tuples

C'est un procédé permettant de décrire formellement l'information que l'on veut extraire, d'une façon non procédurale, et sans spécifier comment on va opérer. Une consultation s'exprime par :

$$\{t \mid P(t)\}$$

soit : rechercher l'ensemble t des tuples vérifiant le prédicat P. Ce dernier est composé de formules atomiques reliées entre elles par des opérateurs logiques (Et, Ou, Non), et dont les variables peuvent être éventuellement quantifiées par $\forall$ et $\exists$ .

Les variables peuvent être des tuples pris globalement, ou réduits à leurs champs significatifs. La recherche prend alors la forme suivante :

$$\{\mathrm{u(r)} \mid (\exists \mathrm{t1} \in \mathrm{R1}) \wedge \ \ldots \ \wedge \ (\exists \mathrm{tn} \in \mathrm{Rn}) \ \wedge \ \mathrm{u}[1] = \mathrm{t_i}[\mathrm{j_1}] \ \wedge \ \ldots \ \wedge \ \mathrm{u}[\mathrm{r}] = \mathrm{t_k}[\mathrm{j_r}] \wedge \Psi\}$$

Soit : trouver une expression u composée de r parties telles qu'il existe un tuple t1 de R1, ...un tuple tn de Rn, que la première partie de u peut être identifiée avec la valeur du $\mathrm{j_1^{ème}}$ attribut de Ri, ...que la $\mathrm{r^{ième}}$ partie de u soit identifiée avec la valeur du $\mathrm{j^{ème}}$ attribut de Rk et qu'une expression $\Psi$ soit aussi vérifiée.

Voici des exemples de requêtes construites sur la base dont le schéma est :
`EMP(num, nom, poste, salaire, service)`
`DEPT(service, batiment)`

- *Liste et localisation des services* :

$$\{\mathrm{D.service}, \mathrm{D.batiment} \mid \mathrm{dept(D)}\}$$

- *Pour chaque service, donner le nom des employés* :

$$\{\mathrm{D.service}, \mathrm{E.nom} \mid \mathrm{Dept(D)} \wedge \mathrm{Emp(E)} \wedge (\mathrm{D.service} = \mathrm{E.service})\}$$

- *Nom des directeurs gagnant plus de 30000 F* :

$$\{\mathrm{E.nom} \mid \mathrm{EMP(E)} \wedge (\mathrm{E.poste} =' \mathrm{directeur}') \wedge (\mathrm{E.salaire} > 30000)\}$$

## 2.2 QUEL

Créé par `INGRES` en 1976 à l'université de Californie, ce langage correspond à une mise en œuvre de l'algèbre relationnelle de tuples.

Ainsi, des expressions comme $(\exists t \in R)$ sont traduites par :

`Range of t is R.`

La recherche d'information par :

`RETRIEVE expression WHERE predicat;`

Les éléments du langage, malgré la volonté de convivialité affichée par ses concepteurs, restent calqués sur l'algèbre. On peut, à titre d'illustration, proposer les exemples suivants :

Equijointure de R et S :

```
Range of r is R;
Range of s is S;
Append to T(C1=r.A1, C2=r.A2, C3=s.B2, C4=s.B3) WHERE r.A1=s.B1
```

Autre exemple, *Nom et salaire des directeurs touchant plus de 30000F :*

```
RETRIEVE E.nom, E.sal WHERE (E.poste='directeur') AND (E.sal > 3000)
```

## 2.3 Calcul relationnel de domaines

Le calcul relationnel de domaines fait appel à des variables prenant leurs valeurs sur le domaine d'un attribut plutôt que sur un tuple entier. Les expressions dans ce mode de calcul sont de la forme :

$$\{(a1, a2, \ldots, an) \mid P(a1, a2, \ldots, an)\}$$

où les $a_i$ représentent les variables d'un domaine et où P est un prédicat composé de formules atomiques construites sur ces mêmes variables.

La position des variables joue un rôle essentiel dans le mécanisme de recherche qui doit identifier les champs concernés au sein de la relation. L'ordre des variables de domaine doit donc correspondre au schéma des relations explorées.

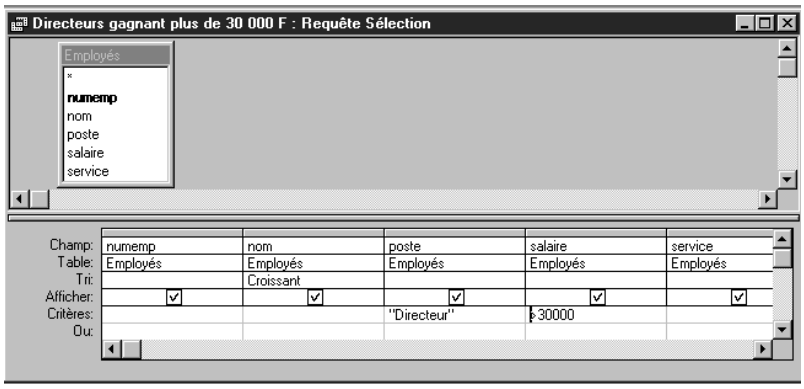
Ainsi, les requêtes proposées en exemple ci-dessus sont traduites par :

- $\{\mathrm{x}, \mathrm{y} \mid \mathrm{DEPT}(\mathrm{service} : \mathrm{x}, \mathrm{batiment} : \mathrm{y})\}$
- $\{\mathrm{x}, \mathrm{y} \mid \mathrm{DEPT}(\mathrm{service} : \mathrm{x}) \wedge \mathrm{EMP}(\mathrm{nom} : \mathrm{y}, \mathrm{service} : \mathrm{z}) \wedge (\mathrm{x} = \mathrm{z})\}$
- $\{\mathrm{x}, \mathrm{y}, \mathrm{z} \mid (\mathrm{EMP} : \mathrm{nom} : \mathrm{x}, \mathrm{poste} : \mathrm{y}, \mathrm{salaire}; \mathrm{z}) \wedge (\mathrm{y} =' \mathrm{directeur}') \wedge (\mathrm{z} > 30000)\}$

## 2.4 QBE

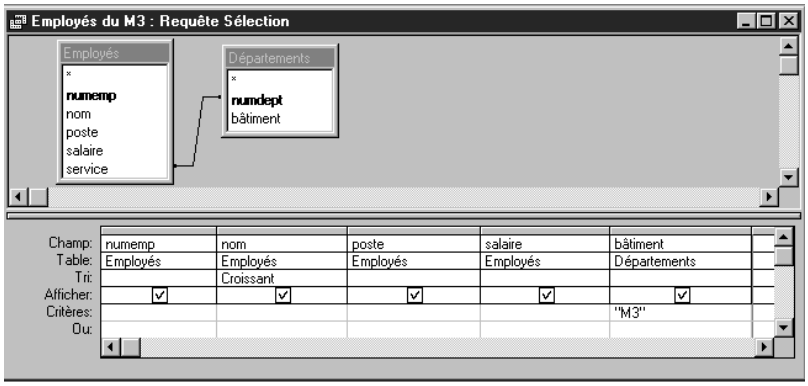
*Query by exemple* (`QBE`) date de 1977 et a été mis au point par Zloff dans les laboratoires d'IBM. Il s'agit d'un interface visuel permettant d'exprimer les concepts du calcul relationnel de domaines. L'utilisateur n' a pas besoin d'être informaticien, encore moins d'être un spécialiste des bases de données. Il demande graphiquement l'affichage des champs qu'il désire visualiser en filtrant les données par des expressions logiques élémentaires. Les jointures sont prises en compte automatiquement lorsque les attributs concernés sont repérés par des variables soulignées. Les recherches par identification ne réclament que la frappe de la valeur dans le champ concerné. Seules les autres comparaisons nécessitent une expression logique complète.

Voici comment `Access` présente l'utilisation de ce langage :




QBE affiche le schéma d'une relation et permet la prise en compte de critères d'interrogation. L'exemple ci-dessus correspond à la recherche des directeurs gagnant plus de 30000F. Les noms seront affichés dans l'ordre alphabétique. Il est possible aussi de demander l'affichage d'expressions calculées, éventuellement renommées par un alias, qui figureront dans la grille d'interrogation.

Dans l'exemple qui suit, on peut voir que les jointures sont automatiquement réalisées dès lors qu'on a indiqué au système les associations à prendre en compte. Il s'agit de donner les noms des employés du bâtiment M3 :




# 3 Extensions

Les limitations des langages de manipulation de bases de données ont donné lieu à toutes sortes de travaux permettant de s'en affranchir. Comme toujours, apparaissent d'abord les offres commerciales destinées à donner une avance à un éditeur sur la concurrence. Vient ensuite la norme qui s'efforce d'entériner le dénominateur commun ou les avancées remarquables.

Deux approches ont vu le jour pour tenter de donner à un langage de type SQL une souplesse propre aux langages procéduraux.

– La création d'un langage adapté par extension d'un langage procédural.
– L'intégration dans un langage donné en assurant une interface adéquate.

La première approche a donné le jour à `PL/SQL` à la suite de bien d'autres tentatives aujourd'hui obsolètes. La seconde, initialisée par des produits tels que `Pro-C`, est bien représentée actuellement par par `PHP`. Cette démarche a abouti à la conception d'`API` dont le modèle est `ODBC` (Microsoft) telle `JDBC` intégrée à `JAVA`.

On trouvera un descriptif succint de PL/SQL dans le poly de TP : *TP de bases de données ; guide de l'utilisateur*

## 3.1 Pro-C et les requêtes dynamiques

Ce produit correspond à une approche par intégration de SQL dans le langage C. On été aussi commercialisés : PRO-BASIC, PRO-COBOL, PRO-PL1, PRO-PASCAL et PRO-ADA.

le principe général d'utilisation consiste à :

– Ecrire un programme en PRO-C

– Le soumettre à un préprocesseur de macros (précompilateur) qui fournit un fichier C. Les requêtes SQL sont seules analysées, transformées en appels de fonctions en bibliothèque et en opérations de gestion de structures C de façon à assurer l'interface avec les données de la base.
– Compiler le programme obtenu avec un compilateur C classique et réaliser l'édition des liens.

Les instructions SQL sont préfixées par la macro `EXEC SQL` dont elles constituent le paramètre. Les outils permettant la traduction sont fournis par un fichier `SQLCA.H` qui initialise une structure complexe de communication entre le programme et le SGBD.

Le programme PRO-C est découpé en deux parties : une zone de déclarations et une zone d'instructions.

### 3.1.1 Déclarations

Elles sont encadrées par les instructions EXEC SQL BEGIN DECLARE SECTION et EXEC SQL EN DECLARE SECTION. Exemple :

```
EXEC SQL BEGIN DECLARE SECTION
  int numero;
  char nom[15];
  char tnoms[20][15];
  float salaire;
EXEC SQL END DECLARE SECTION.
```

Les variables ont des types conformes aux types de base de SQL. Un cas particulier est posé par le type VARCHAR, qui est représenté de façon interne par un enregistrement à deux champs :

```
Exec SQL begin declare section
  VARCHAR nom[30];
Exec SQL end declare section
```

Le précompilateur réalise l'expansion de la façon suivante :

```
struct{
  unsigned short int len;
  unsigned char arr[30];
       }nom;
```

Il faut donc lors de l'utilisation d'un VARCHAR, manipuler explicitement les champs correspondant en utilisant des fonctions telles que `strcpy`, `strcat`, `strlen`, ...comme l'illustre l'exemple ci-après.

Certaines variables ont un statut particulier. On les désigne sous le nom de *variables-hôtes (host variables)*. Elles sont déclarées dans la `declare section`, et sont utilisées dans les requêtes SQL sur la base d'une syntaxe étendue, préfixées par le symbole ':'

### 3.1.2 Requêtes dynamiques

Il est possible de définir dynamiquement le texte des requêtes à exécuter, en fonction de l'état des données et du déroulement du programme. Ces requêtes dynamique utilisent un curseur associé à une variable destinée à contenir le texte de la requête. Par rapport au mécanisme classique de gestion d'un curseur, la seule nouveauté réside dans une instruction `prepare` qui réalise l'association.

Dans l'exemple suivant, on veut réaliser une synthèse des réponses à un questionnaire. Les réponses sont stockées dans une table QUESTIONS, de 5 colonnes $Q_1$ jusqu'à $Q_5$. Chaque réponse est codée par des entiers de 1 à 3, et le programme compte le nombre de réponses identiques.

```
#include <stdio.h>
#include <stdlib.h>
EXEC SQL begin declare section
  VARCHAR uid[20];
  VARCHAR pswd[20];
  VARCHAR requete[120];
  int total;
EXEC SQL end declare section
char numq[5]={"1","2","3","4","5"};
char valeur[1];
int i,j;
EXEC SQL INCLUDE SQLCA;
main(){
  strcpy(uid.arr,"monNom");
  uid.len=strlen(uid.arr);
  strcpy(pswd.arr,"monMotDePasse");
  pswd.len=strlen(pswd.arr);
  EXEC SQL WHENEVER SQLERROR STOP;
  EXEC SQL CONNECT :uid IDENTIFIED BY :pswd;
  printf("Connexion a Oracle reussie\n");
  for(i=0;i<5;i++){
    strcpy(requete.arr,"SELECT COUNT(Q");
    strcat(requete.arr,numq[i]);
    strcat(requete.arr,") FROM QUESTIONS WHERE Q");
    strcat(requete.arr,numq[i]);
    strcat(requete.arr," = ");
    for(j=1;j<=3;j++){
      itoa(j,valeur,10);     /* conversion en base 10 d'un entier en ASCII */
      strcat(requete.arr,valeur);
      requete.len=strlen(requete.arr);
      EXEC SQL PREPARE S FROM :requete;
      EXEC SQL DECLARE C CURSEUR FOR S;
      EXEC SQL OPEN C;
      EXEC SQL WHENEVER NOT FOUND GOTO fin;
      EXEC SQL FETCH C INTO :total;
    }
  }
  fin:
      EXEC SQL CLOSE C;
      printf("reponse %d a la question %d : %s \n",j,i,total);}}
  EXEC SQL COMMIT WORK RELEASE;
  exit(0); }
```

# 4 Les API

Les API (Application Programming Interface) sont décrites dans la norme SQL2. L'idée est de donner à l'utilisateur des librairies de fonctions assurant l'interface entre le SGBD et l'application ; c'est ce qui est réalisé par des bibiliothèques propres à certains produits (`PRO-C` d'Oracle), avec l'inconvénient d'avoir des outils propriétaires. Ici, la librairie est normalisée et documentée ; elle peut être invoquée par n'importe quel langage, quel qu'en soit le compilateur. Le API sont largement utilisées aujourd'hui en mode client/serveur.

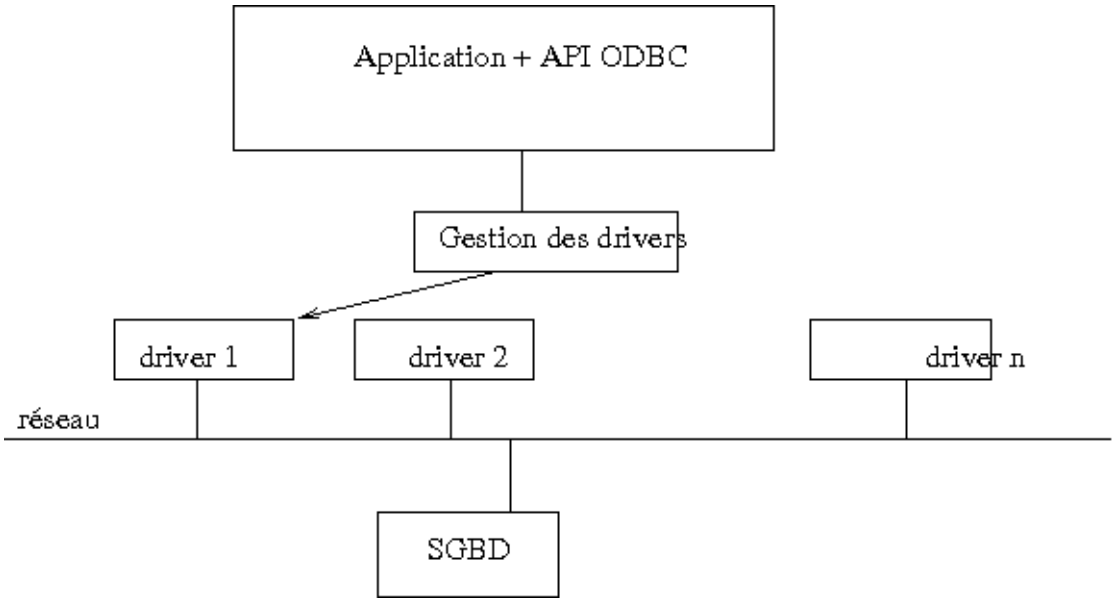
C'est en 1992 que Microsoft a proposé `ODBC` (Open database Connectivity), et devant son succès, ses concurrents ont tous depuis proposé des offres similaires.

## 4.1 Généralités

Une API permet la gestion des curseurs, les requêtes dynamiques, les accès concurrents ; Elle remplace l'intégration dans un langage procédural. En autorisant la notion de trains de requêtes, elle réduit le traffic réseau en mode Client/Serveur.

ODBC est maintenant très répandu et a été conçu en couches : le gestionnaire de drivers et les drivers sont séparés. Il suffit du driver ODBC adéquat pour rendre les requêtes compréhensibles par ce dernier. Comme ce produit est devenu une norme de fait, les éditeurs, en écrivant eux-même les drivers pour chaque nouvelle version, assurent la compatibilité de leur produit avec les anciennes applications ; cette technique est beaucoup plus souple, car il est nettement plus facile de modifier un driver que le noyau du SGBD. Quant au développeur, il dispose d'un kit de dévelopement : SDK (Software Devlopement Kit).

Devant le succès d'ODBC, tous les éditeurs proposent aujourd'hui un driver ODBC pour leur produit.




# 5 JDBC

JDBC signifie *Java Data Base Connectivity* et constitue la réponse de `SUN` au développement fulgurant d'`ODBC` de `MicroSoft`. L'un comme l'autre sont d'ailleurs devenus des standards de la communication entre un programme et une base de donnée.

JDBC est, comme ODBC, une API (`Application Interface Programming`), c'est à dire une bibliothèque offrant des classes et des interfaces pour se connecter et exploiter des SGBD à partir de programmes Java. Il est ainsi possible d'exploiter les SGBD en Client/Serveur ou en utilisant le *Web*.

Documentation : `http://java.sun.com/javase/6/docs/api/`

## 5.1 Programmer avec JDBC

### 5.1.1 Liste des principales classes et interfaces

| Interfaces | Description |
|---|---|
| Driver | Renvoie une connexion utilisée par le DriverManager |
| Connection | Connexion à une base |
| Statement | lié à un ordre SQL |
| PreparedStatement | lié à un ordre SQL paramétré |
| CallableStatement | lié à une procédure stockée sur le serveur |
| ResultSet | lignes issues d'un SELECT |
| ResultSetMetaData | description de lignes du SELECT |
| DatabaseMataData | informations du dictionnaire des données. |

| Classes | Description |
|---|---|
| DriverManager | gère les drivers, lance les connexions |
| Date | dates SQL |
| Time | h mn s SQL |
| TimeStamp | estampille |
| Types | descriptions des types SQL |
| DriverPropertyInfo | informations pour la connexion |

Il faut en premier lieu disposer des classes (*drivers*) qui implémentent les interfaces de JDBC. Ces classes doivent fournir la méthode `connect()` qui renvoie une instance de la classe qui implémente l'interface `Connection`. Les requêtes sont lancées ensuite en créant les instances de classes qui implémentent l'interface `Statement`.

Le chargement de plusieurs drivers permet de travailler avec plusieurs SGBD.

Quand une classe correspondant à l'interface *Driver* est chargée en mémoire, une instance de cette dernière est créée et le driver est enregistré par le *DriverManager*. Lors d'une tentative de connexion, celui-ci tente d'effectuer la connexion avec tous les drivers enregistrés et sélectionne le premier qui convient dans l'ordre des déclarations. De ce fait, l'ordre d'instanciation des différents drivers a une importance capitale.

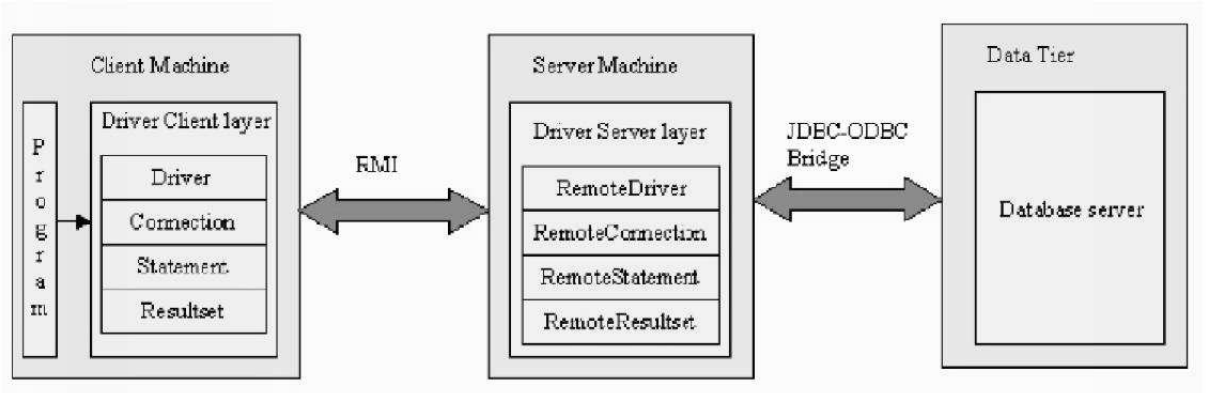
Une application Java peut travailler avec tous ces types de drivers.

Il est possible, pour accéder à un SGBD via le Web, d'utiliser des *servlets* qui sont des applications Java libérées des contraintes existant sur les appliquettes (*applets*). Ces *servlets* sont installées côté serveur et doivent être gérées par un serveur web spécifique (norme *apache*), tel que `tomcat`. Elles peuvent générer des pages Web contenant des données récupérées de la base et constitue une variante dynamique des programmes CGI.

### 5.1.2 Modèle de connexion

L'architecture la plus fréquente aujourd'hui correspond à ce qu'on appelle un **modèle 3-tiers** : un serveur web *middleware* ($2^{nd}$ tiers) dialoguant grâce à JDBC avec le SGBD ($3^{eme}$ tiers), sert d'interface avec l'application Java (sur le client : $1^{er}$ tiers ). Ce modèle assure une plus grande sécurité du SGBD et une plus grande souplesse dans la programmation. Il permet entre autre de communiquer avec non pas un, mais plusieurs SGBD. C'est ce modèle qui est utilisé pour le traitement des `servlets`.

La figure suivante illustre le principe d'utilisation de JDBC dans le cas d'une architecture 3/tiers :




### 5.1.3 JDBC en pratique :

Sous Windows, le plus simple est d'utiliser `NetBeans` qui est un outil intégré de développement et encapsule toutes les variables d'environnement ainsi que l'accès au compilateur et à la machine JAVA.

Sinon, il faut (cf. figure page suivante) donner une visibilité aux classes implémentant JDBC en définissant des variables d'environnements. Pour cela, ouvrir la fenêtre des paramètres, choisir `system`, puis `Avancé` et enfin cliquer sur `Variables d'environnement`.

Dans la partie *utilisateurs*, définir `JAVA_HOME` avec, par exemple, si on a placé le répertoire du jdk sous la racine, la valeur : `C:\jdkk1.5.0_06\`

Dans la partie *Système*, définir `CLASSPATH` avec la valeur :
`C:\jdk1.5.0_06\lib\dt.jar;C:\jdk1.5.0_06\lib\tools.jar;C:\jdk1.5.0_06\lib\mysql-connector-java-3.1.7-bin.jar`
si on a placé dans le répertoire lib du jdk le driver mysql.

Définir enfin `PATH` avec la valeur : `%PATH%;C:\jdk1.5.0_06\bin`

Ceci étant réalisé, on ouvre l'*invite de commande*, et on lance la compilation avec la commande : `javac MaClasse.java` et la machine java (exécution) avec `java MaClasse`.

Au niveau du programme Java, la démarche, assez répétitive, consiste à :

1. Importer le paquetage `java.sql`
2. Charger en mémoire la classe du driver, par exemple avec l'invocation suivante du Class.forName :
   `Class.forName("com.mysql.jdbc.Driver");`
   Puis :
3. Ouvrir une connexion : `Connection con = java.sql.DriverManager.getConnection(...)`
4. Créer des instructions SQL en exploitant l'une des classes implémentant : `Statement`, `PreparedStatement`, `CallableStatement`.
5. Lancer les requêtes avec, selon le cas, une des méthodes de Statement : `executeQuery()`, `executeUpdate()` ou `execute()`.
6. Fermer la connexion : `con.close()`


### Ouvrir une connexion

Une connexion à un SGBD est représentée par une instance de la classe implémentant l'interface `Connection`. On l'obtient en retour de la méthode *static* `getConnection()` de la classe `DriverManager`. On doit passer à cette méthode l'URL de la base de donnée. Le gestionnaire de driver essaye tous les drivers enregistrés lors de leur chargement par `Class.forName()` jusqu'à ce qu'il en trouve un qui lui permette de se connecter à la base.

Une URL de base de donnée est de la forme : *jdbc :sous-protocole :base de donnée*

Par exemple, avec un pont JDBC-ODBC on pourrait avoir : `jdbc:odbc:base` En fait, la syntaxe de l'URL dépend du driver utilisé. Il faut se reporter à la documentation du driver.

Pour les bases de données suivantes, les pilotes sont :

| BD | pilote | URL |
|---|---|---|
| Oracle | `oracle.jdbc.driver.OracleDriver` | `jdbc:oracle:thin:serveur:1521:chaine\_de\_connexion` |
| MySQL | `com.mysql.jdbc.Driver` | `jdbc:mysql:serveur/nomDeLaBase` |
| postgreSQL | `org.postgresql.Driver` | `jdbc:postgresql:serveur/nomDeLaBase` |

Ainsi avec mysql, pour la base *essai* située sur un serveur local, pour un utilisateur dont les login et mot de passe seraient contenus par les variables `uid` et `pswd`, on aurait :

```
static final String url = "jdbc:mysql://localhost/essai";
Connection con = DriverManager.getConnection(url, uid, pswd);
```

Le pilote MySQL peut être téléchargé à partir de mysql.com. Une fois l'installation faite (par exemple dans le répertoire `lib` d'EasyPHP), la variable d'environnement CLASSPATH doit naturellement lui donner sa visibilité.

La séquence complète de connexion pour ce SGBD sera alors, si l'utilisateur est le DBA `root` (sans mot de passe) :

```
try{
    Class.forName("com.mysql.jdbc.Driver");
    java.sql.Connection con = java.sql.DriverManager.getConnection(
                "jdbc:mysql://localhost/essai","root","");
} catch(Exception e){
    System.out.println("Pb de driver : " + e.getMessage()););
}
```

#### Instructions SQL simples

Les ordres SQL sont représentés par des instances de la classe implémentant l'interface `Statement`. La méthode à appeler est différente suivant la nature de la requête ; la consultation utilise `executeQuery()`, la modification `executeUpdate()`.

- **Interrogation**
  La requête (Select) est passée à la méthode `excuteQuery(String)`. Celle-ci renvoit une instance de la classe `ResultSet`, qui est une représentation des lignes retournées par le Select. Ces dernières sont parcourues par la méthode `next()` :

  ```
  Statement stmt = connexion.createStatement();
  ResultSet rset = stmt.executeQuery("select nom from emp");
  while (rset.next() )
    System.out.println (rset.getString(1));  // getString(1) récupère la première colonne
  stmt.close();
  ```
- **Récupération des données**
  La classe `ResultSet` fournit des méthodes *getXXX(int numéroColonne)* ou *getXXX(String nomColonne)* pour récupérer dans le code Java les valeurs de colonnes des lignes renvoyées par l'interrogation. Par exemple, `getString()` renvoit une instance de la classe `String`.
  Il existe des possibilités de conversion automatique, mais si celle-ci est impossible, la méthode lance une `SQLException`.
  Oracle n'utilise pratiquement que le type `number` avec lequel on peut utiliser `getShort()`, `getInt()`, `getLong()`, `getFloat()`, `getDouble()`.

| Type SQL | Méthode Java conseillée |
|---|---|
| CHAR/CHAR(n) | getString() |
| VARCHAR(n)/VARCHAR2(n) | getString() |
| NUMBER | getXXX() |
| DATE | getDate() |

  **Rem** : quand on récupère un *CHAR(n)* dans un objet de la classe `String`, la valeur est complétée par des espaces.
- **Valeur NULL**
  Celle-ci est repérée par la méthode `wasNull()` renvoyant un booléen . Dans l'exemple qui suit, on imprime des noms d'employés associé avec une éventuelle commission lorsque celle-ci existe :

  ```
  Statement stmt = connexion.CreateStatement();
  ResultSet rset = stmt.executeQuery("select nom, commission from emp");
  ```

```
Float commission;
while (rset.next()) {
  nom = rset.getString(1);
  commission = rset.getFloat(2);
  if (rset.wasNull())
    System.out.println(nom + " pas de commission");
  else
    System.out.println(nom + " avec " +  commission + " euros de commission");
}
```

– **Modification de données**

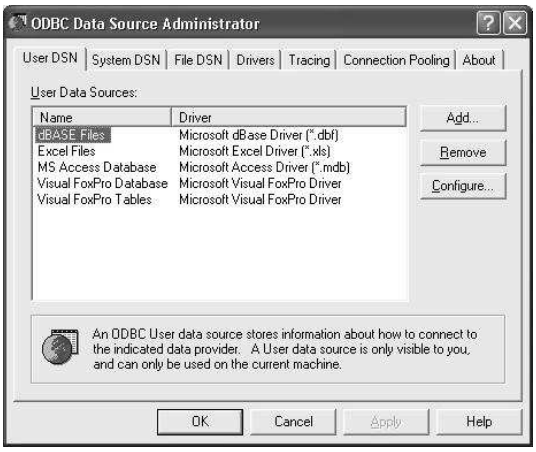
```
Statement stmt = connexion.createStatement();
String ville = new String ("Villeneuve d''Ascq");
int nbLignesModifiees = stmt.executeUpdate("insert
   into vendeur values(15,'toto',"'" + ville + "'"));
stmt.close();
```

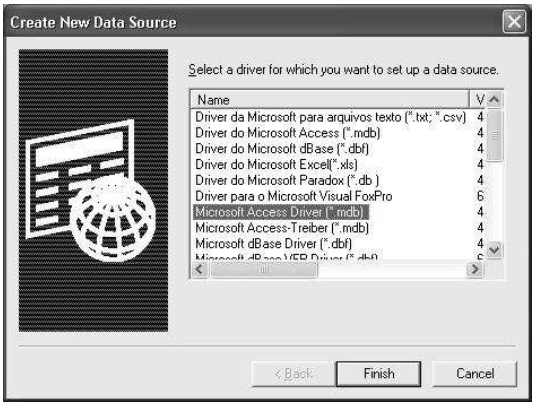
Attention à bien mettre deux apostrophes simples entre le `d` et le nom `Ascq` de la ville.
La méthode `executeUpdate()` doit toujours être utilisée pour lancer des ordres SQL ne renvoyant aucun résultat, comme toutes les commandes d'administration : ( Create Table, Grant, ...). Elle retourne un code d'exécution ou le nombre d elignes affectées selon la nature de la requête SQL.
La méthode `execute`, quant à elle, peut remplacer avantageusement les deux précédentes.

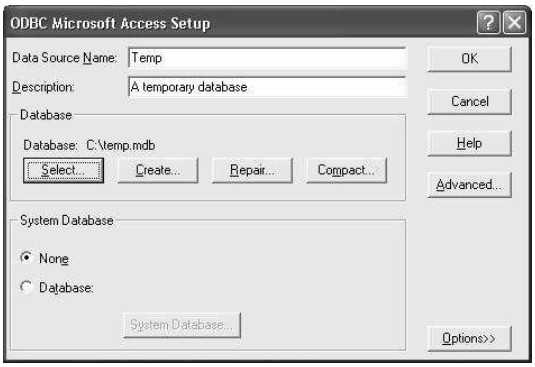
## 5.2 Exemples d'applications JDBC

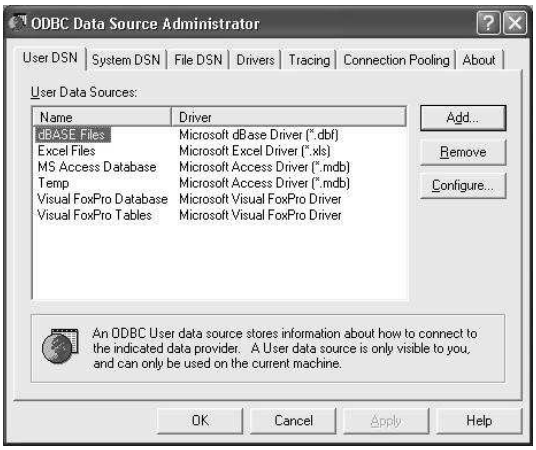
Les exemples qui suivent correspondent à trois applications tournant sous Windows en réalisant un pont ODBC/JDBC avec une base `Access`. Ces trois applications permettent de : créer une table, y insérer des données, puis lire ces dernières.

Ils supposent la création préalable d'un driver ODBC, réalisée en allant chercher le panneau de configuration dans les paramètres, puis en cliquant sur l'icône *ODBC 32 bits*. On clique sur le bouton `Add` (fig.a), on choisit le type de driver (`Microsoft Access Driver`). La fenêtre qui s'ouvre alors (fig. c) permet d'associer au nom symbolique d'un driver (ici *Temp* dans l'exemple) le type et le chemin d'accès à la base de données. La figure d montre ce driver en place parmi les autres :






## 5.3 Manipulation d'une table Access

```
/* Essai.java : Essais avec  la table  matable*/
import java.sql.*;

 public class Essai{
  public static void main(String args[]){

   String url = "jdbc:odbc:temp;
  //le lien  ODBC a pour nom symbolique temp

   Connection con;
   Statement stmt;
   String requete;

   try {
     Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
   }
   catch(java.lang.ClassNotFoundException e){
    System.err.print("ClassNotFoundException : ");
    System.err.println(e.getMessage());
   }
   try{
    con=DriverManager.getConnection(url,"","");
    stmt=con.createStatement();

    // Création de la table
    requete = "Create table matable(Nom varchar(10), prenom varchar(10),
            adresse varchar(30))";
    stmt.executeUpdate(createString);

    // remplissage de la table
    stmt.executeUpdate("Insert into matable values ('Toto','Paul','25 rue DOS', 'Lille')");
    stmt.executeUpdate("Insert into matable  values('Truc','Bob','32 Av Linux', 'Roubaix')");
    stmt.executeUpdate("Insert into matable values('Machin','Momo','10  Bd OS','Tourcoing')");

    // interrogation de la table
    requete = "select nom, prenom, adresse from matable";
    ResultSet rs = stmt.executeQuery(query);
    System.out.println("Liste des personnes");
    while (rs.next()){
     String n = rs.getString("nom");
     String p = rs.getString("prenom");
     String a = rs.getString("adresse");
     System.out.println(n+" "+p+" "+a);
```

```
    }

    stmt.close();
    con.close();
    }
    catch(SQLException ex){
     System.err.println
        ("SQLException : "+ex.getMessage());
}}}
```

## 5.4 Accès à la métabase

La méthode `con.getMetaData()` de l'objet de connexion `con` permet d'obtenir les méta-données, sous la forme d'un objet typé par : `DatabaseMetaData`.

Soit `dmd` un tel objet. La méthode

`getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)`

fournit alors, sous la forme d'un objet `ResultSet`, la liste des objets de la base correspondant au types passés en paramètre. Par exemple l'objet : `String les_types={"TABLE" ,"VIEW"};` permet d'accéder à la liste des tables et des vues de l'utilisateur. Le catalogue (dictionnaire des données) peut être vide (`null`) ou obtenu par `con.getCatalog()`. Le schéma sera une chaîne vide sous Postgres (pas de schéma) ou défini par le nom de l'utilisateur en majuscules (`this.login.toUpperCase()` sous Oracle)

Ainsi, sous Oracle, la séquence :

`ResultSet les_tables = dmd.getTables(con.getCatalog(),null,"%",les_types)` va placer dans un `ResultSet` la liste des tables et vues.

`les_tables.getString("table_name")` permet d'obtenir le nom de l'objet (table ou vue),

et `les_tables.getString("table_type")` indiquera s'il s'agit d'une table ou d'une vue.

- Si, après exécution d'une requête d'interrogation, on souhaite afficher, en préalable aux données récupérées, un chapeau contenant les noms des colonnes, on peut procéder de la manière suivante :
  L'interrogation d'une table (ou vue) place le résultat dans un objet `ResultSet`. Les méta-données de cet objet (appelons-le `rs`) sont accessibles via la classe `ResultSetMetaData`. Par exemple :
  `ResultSetMetaData rsmd=rs.getMetaData()`
  Le nombre de colonnes de la table peut être obtenu à partir de : `rsmd.getMetaData().getColumnCount()`, et le nom de la colonne numéro `i` par : `rsmd.getMetaData().getColumnName(i)`.
- On peut aussi avoir accès directement au schéma relationnel de chaque table de la façon suivante :
  La méthode

`getColumns(String catalog,String schemaPattern,String tableNamePattern,String columnNamePattern)`

  de la classe `DataBaseMetaData` fournit un résultat de type `ResultSet`.
  `ResultSet les_colonnes=dmd.getColumns(null,le_schema,nomTable,"%");`
  En explorant les colonnes retournées par la méthode `next`, on peut invoquer :
  `les_colonnes.getString("column_name");` qui donne les noms des colonnes
  `les_colonnes.getInt("data_type");` qui retourne le code du type de la colonne
  `les_colonnes.getInt("column_size");` qui retourne sa longueur éventuelle
  `les_colonnes.getString("is_nullable");` qui indique si la colonne peut contenir ou non des valeurs nulles.
  Les méthodes qui suivent sont des méthodes de `DataBaseMetaData` et retournent un objet de type `ResultSet` :
  `getPrimaryKeys(String catalog, String schema, String table)` retourne la clé primaire
  `getExportedKeys(String catalog, String schema, String table)` retourne les attributs référencés
  `getImportedKeys(String catalog, String schema, String table)` retourne les attributs référençants.
  Exemple :

```
ResultSet les_cles=dmd.getPrimaryKeys(null, le_schema, nomTable);
if (les_cles)==null){System.out.println("Pas de clé primaire définie");}
else{
   System.out.println("Clé primaire : ");
  while (les_cles.next(){
    System.out.println(les_cles.getString("COLUMN_NAME");
  }
}
```

En ce qui concerne les types SQL retournées par l'intermédiaire des méta données, ce sont des codes entiers définis dans : `java.sql.Types`. Ces codes portent un nom ; on a les équivalences suivantes :

```
java.sql.Types.CHAR correspond au type CHAR
java.sql.Types.VARCHAR  correspond au type VARCHAR
```

```
java.sql.Types.INTEGER  correspond au type INTEGER
java.sql.Types.FLOAT  correspond au type FLOAT
java.sql.Types.DATE  correspond au type DATE
```

sans donc avoir à se préoccuper de la valeur de leur valeur.

# 6 Servlets

Il existe de nombreux moyens d'accèder via le web à une ou des bases de données. Ces moyens peuvent utiliser des programmes JAVA, mais aussi nombre d'autres langages de programmation (du C au PHP).

Deux procédés populaires et symétriques existent :

- l'écriture de `servlets`, programmes java tournant sur le serveur et offrant ainsi un maximum de sécurité. La servlet correspond avec l'utilisateur en retournant des séquence HTML constituant les pages de réponse à ses requêtes.
- des pages `JSP` qui sont constituées de séquences JAVA encapsulées dans du HTML par l'intermédiaire de marqueurs spécifiques. Le JSP est transformé automatiquement en servlet au niveau du serveur, mais sont contenu est visible par le client et ne permet pas d'assurer une sécurité suffisante.

**Schéma d'une application WEB**

Une application Web :

1. Recueille les données utilisateur
2. Envoie une requête au serveur Web
3. Exécute le programme d'application (serveur d'application/de données)
4. retourne les résultats au navigateur

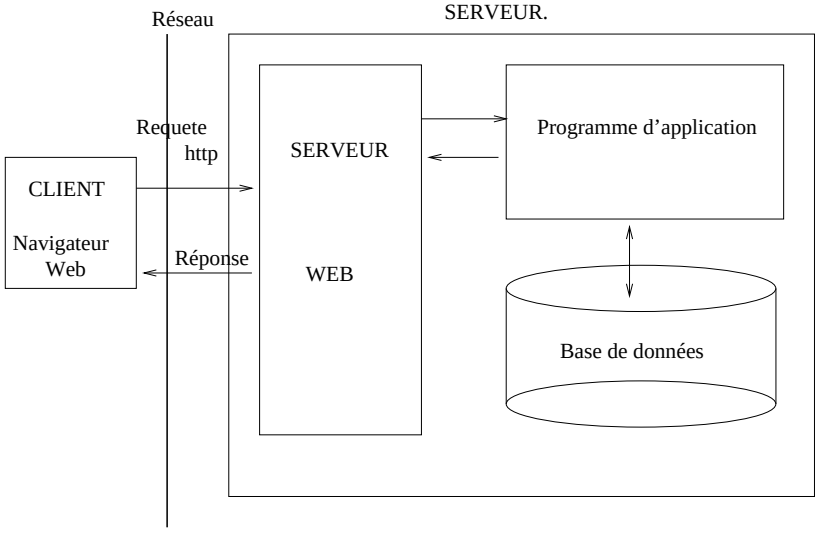
Le traitement des données issues d'un formulaire HTML se fait avec deux méthodes, GET et POST :

- la requête GET :
  - Extrait des informations sur le serveur HTTP
  - utilise la technique CGI pour intégrer les données de formatage à l'URL. Exemple :

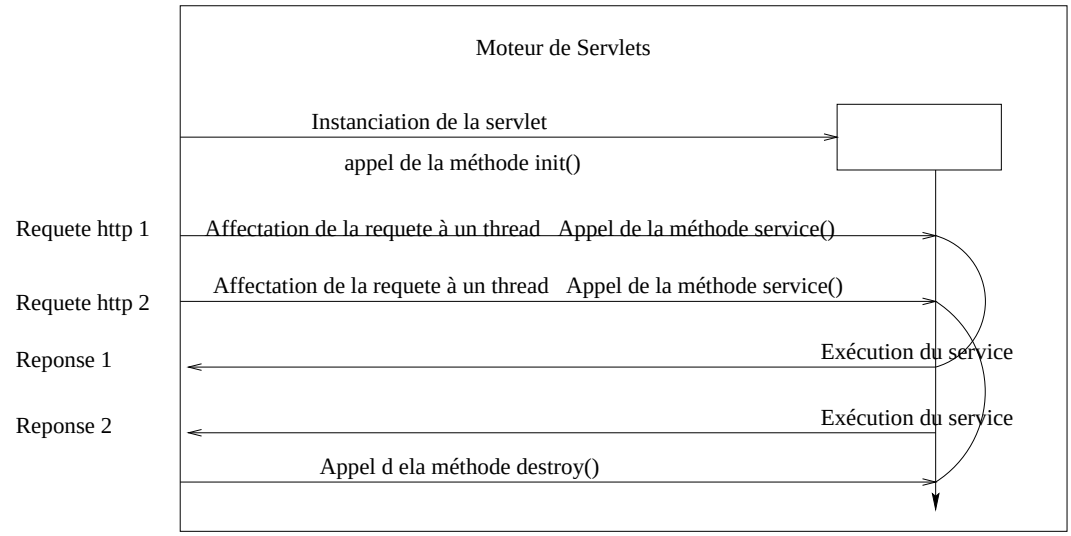
```
http://www.fil.univ-lille1.fr/monAppli?x=3&y=4
```

- la requête POST
  - Modifie des données sur le serveur
  - Les données de la page sont assemblées et envoyées vers le serveur
  - Permet d'exploiter un plus gros volume de données

Le retour des résultats au navigateur se fait de la façon suivante : du côté serveur, le programme précise la nature du contenu(HTML, images, ...) et intègre la réponse dans le flot de sortie. Du côté client, le navigateur définit le type MIME de l'en-tête et affiche des données encapsulées dans du HTML




**Servlets : principes de fonctionnement**




- Une servlet doit implémenter l'interface `javax.servlet.Servlet`. Il faut noter que 'importation de `javax.servlet.*` et de `javax.servlet.http.*` devient inutile avec le `jdk1.3` (il faut importer `java.net.*` à la place), mais redevient indispensable avec le j2sdk1.4
- Trois méthodes doivent être implémentées :
  - `init()` qui initialise la servlet
  - `service()` qui traite les requêtes
  - `destroy()` qui détruit la servlet et libère les ressources

Dans la classe `HttpServlet`, on trouve deux méthodes qui remplacent la méthode `service()` de la classe mère : `doGet()` pour les requêtes HTTP de type GET, et `doPost()` pour les requêtes HTTP de type POST

L'exemple ci-dessous indique comment doit se réaliser une implémentation de `doPost` :

```
public class essai1 extends HttpServlet{
   public void init(HttpServletConfig c)
             throws ServletException{....}

   public void doPost( HttpServletRequest req, HttpServletResponse res)
             throws ServletException, IOException{
       PrinterWriter out=res.getWriter();
       res.setContentType("text/html");
       out.println("<html>");
       ....
   }

  public void destroy(){....}
}
```

Quant à la récupération de paramètres, son principe est illustré par l'implémentation suivante de `doGet` :

```
public void doGet(HttpServletRequest req, HttpServletResponse rep)
      throws ServletException, IOException{
  Enumeration liste = req.getParameterNames();
  String [] valeurs = req.getParameterValues();
  String val1 = req.getParameter(‘‘param1’’);
```

**Exemple : annuaire DESS**

L'exemple qui suit permet de réaliser un annuaire des étudiants du DESS, à partir de deux fichiers : un fichier HTML stocké sur le serveur Web et auquel un client accède par l'intermédiaire de son navigateur, et un fichier java.

La page HTML contient un formulaire permettant de rentrer le nom d'un étudiant. Le programme java prend ce nom en paramètre, interroge la base, et renvoit en réponse un page HTML de présentation des résultats.

**La pageHTML**

```
<HTML>
<HEAD><TITLE> ANNUAIRE TIIR </TITLE></HEAD>
<BODY BGCOLOR="#FFFFFF"<CENTER>
<CENTER><H1>ANNUAIRE DU DESS TIIR </H1></CENTER>
<HR><CENTER>
<H2>Recherche de coordonnées </H2></CENTER>
<P> Tapez le début du nom de la personne recherchée:
<P><FORM METHOD=POST ACTION=http://localhost:8080/examples/servlets/annuaire  method=post>
<INPUT TYPE=TEXT NAME="nom" SIZE=20  MAXLENGTH=30 VALUE="">
<P><INPUT TYPE=SUBMIT NAME="go" VALUE="RECHERCHER">
<INPUT TYPE=RESET NAME="reset" VALUE="ANNULER">
</FORM>
</BODY></HTML>
```

**La Servlet par elle-même**

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Annuaire extends HttpServlet{
 public void doPost( HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException{
    res.setContType("text/html");
    PrintWriter out=res.getWriter();
    out.println("<HEAD><TITLE>Réponse annuaire </TITLE></HEAD><BODY>");
    out.println("<HR>");
    try{
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
      String url ="jdbc:odbc:mabase";
      java.sql.Connection c=DriverManager.getConnection(url,"","");
     java.sql.Statement st = c.createStatement();
     java.sql.ResultSet rs =
       st.executeQuery("Select * from matable where nom like '"
          +req.getParameter("nom"+"*'"));
     rs.next();
     out.println(rs.getString("prenom")+" "+rs.getString("nom") );
   }
   catch (SQLException e){out.println("Cette personne n'existe pas");}
   out.println("<P><A href = annuaire.html> Retour</A>"</P>);
   out.println("</BODY>");
   out.close();
 }
  public String getServletInfo(){
   return "Servlet Annuaire";
  }
```

# 7 INTRODUCTION à PHP

## 7.1 Qu'est-ce que PHP ?

PHP est un langage de script interprété exécuté du côté serveur. La syntaxe du langage est issue de celles du langage C, du Perl et de Java. Elle est de ce fait peu lisible et ce langage s'apparente à un bricolage réalisé sans les apports de la théorie des langages ni pensée directrice. Il a cependant des avantages dans le lot des langages de scripts disponibles :

- La gratuité et la disponibilité du code source (PHP3 est distribué sous licence GNU GPL)
- la possibilité d'inclure le script PHP au sein d'une page HTML
- La simplicité d'interfaçage avec des bases de données (de nombreux SGBD sont supportés, mais le plus utilisé avec ce langage est MySQL.
- L'intégration au sein de nombreux serveurs web (Apache, EasyPHP, Microsoft IIS, ...)

## 7.2 Origines de PHP

Le langage PHP a été mis au point au début d'automne 1994 par Rasmus Lerdorf. Ce langage de script lui permettait de conserver la trace des utilisateurs venant consulter son CV sur son site, en accèdant à une base de données par l'intermédiaire de requêtes SQL. Par la suite, de nombreux internautes lui demandèrent ce programme et Rasmus Lerdorf mit en ligne en 1995 la première version qu'il baptisa Personal Sommaire Page Tools, puis Personal Home Page v1.0.

Etant donné le succès de PHP 1.0, il décida d'améliorer ce langage en y intégrant des structures plus avancées telles que des boucles, des structures conditionnelles, un paquetage permettant d'interpréter les formulaires qu'il avait développé (FI, Form Interpreter) ainsi que le support de mSQL. C'est de cette façon que la version 2 du langage, baptisée pour l'occasion PHP/FI version 2, vit le jour durant l'été 1995. Il fut rapidement utilisé sur de nombreux sites (15000 fin 1996, puis 50000 en milieu d'année 1997). A partir de 1997, Zeev Suraski et Andi Gurmans rejoignèrent Rasmus pour former une équipe de programmeurs afin de mettre au point PHP 3 (Stig Bakken, Shane Caraveo et Jim Winstead les rejoignèrent par la suite). La version 3.0 de PHP fut disponible le 6 juin 1998. A la fin de l'année 1999, une version bêta de PHP, baptisée PHP4 est apparue...

L'intérêt du langage est qu'il est doté de nombreuses fonctions assurant quasiment toutes les fonctionnalités du shell UNIX, et qu'il peut donc se substituer à un shelle, à la manière de `perl` ou `python`. Il permet également de communiquer avec la plupart des SGBD.

## 7.3 SGBD supportés par Php

PHP permet un interfaçage simple avec de nombreux SGBD, parmi lesquels :

Informix, MySQL, PostgreSQL, Oracle, Sybase

Cet interfaçage est réalisé par l'intermédiaire de fonctions spécialisées, intégrées au langage, et qui correspondent aux fonctions que l'on trouverait dans une API indépendante.

## 7.4 PHP est interprété par le serveur

Un script PHP est un simple fichier texte contenant des instructions écrites à l'aide de caractères ASCII 7 bits incluses dans un code HTML à l'aide de balises spéciales et stocké sur le serveur. Ce fichier doit avoir l'extension `.php` pour pouvoir être interprété par le serveur !

Un script PHP est stocké sur et interprété par le serveur, les utilisateurs ne peuvent donc pas voir le source.

## 7.5 Implantation au sein du code HTML

Le script peut être interprété par le serveur de deux façon sdifférentes

- Le fichier contenant le code a une extension en .php et ne contient que du PHP. Il peut générer, à l'aide des procédures d'impression, du code HTML destiné au client.
- Le fichier est un fichier HTML. Le code PHP doit alors être délimité par les balises < ?php et ?>

Voici un exemple élémentaire :

```
<html>
<head><title>Exemple</title></head>
<body>
 <?php
   echo "Hello world";
 ?>
</body>
```

```
</html>
```

## 7.6 L'interprétation du code

Un code PHP (celui compris entre les délimiteurs < ?php et ?> dans un fichier HTML ) est un ensemble d'instructions se terminant chacune par un point-virgule (comme en langage C). Lorsque le code est interprété, les espaces, retours chariot et tabulation ne sont pas pris en compte par le serveur.

Les commentaires sont délimités par /* et */ Un commentaire sera donc noté de la façon suivante :

```
/* Voici un commentaire! */
```

Les commentaires peuvent être écrits sur plusieurs lignes, mais ne peuvent être imbriqués.

Il est possible également de mettre toute la fin d'une ligne en commentaire en utilisant le double slash (//).

Le langage PHP est par exemple sensible à la casse (en anglais case sensitive). Toutefois, cette règle ne s'applique pas aux fonctions, les spécifications du langage PHP précisent que la fonction print peut être appelée print(), Print() ou PRINT(). Enfin, toute instruction se termine par un point-virgule.

## 7.7 Introduction à EasyPHP

Afin de faire fonctionner PHP, il est nécessaire de disposer d'un serveur APACHE intégrant un module d'interprétation. C'est le cas d'`EasyPHP` qui intègre de plus `MySQL`.

EasyPHP est disponible sur le site : `www.easyphp.org`

## 7.8 Récupération de données

PHP rend très simple la récupération de données envoyées par l'intermédiaire de formulaires HTML.

```
<FORM Method="POST" Action="test.php">
Nom :           <INPUT type=text size=20 name=nom><BR>
Prénom :        <INPUT type=text size=20 name=prenom><BR>
Age :           <INPUT type=text size=2 name=age><BR>
<INPUT type=submit value=Envoyer>
</FORM>
```

Lorsque l'on soumet un formulaire à un fichier Php, toutes les données du formulaires lui sont passées en tant que variables, dont les noms correspondent à ceux des champs du formulaires, et ce par l'intermédiaire d'une table nommée `$_POST[]` ou `$_GET[]`, selon la méthode de transfert invoquée.

```
// fichier test.php
<?php
$nom=$_POST['nom'];
$prenom=$_POST['prenom'];
$age=$_POST['age'];

 if (($nom=="")||($prenom=="")||($age==""))
 {
  if($nom=="") print("Veuillez saisir le nom de l'utilisateur<BR>\n");
  if($nom=="") print("Veuillez saisir le prénom de l'utilisateur<BR>\n");
  if($nom=="") print("Veuillez saisir l'age de l'utilisateur<BR>\n");
 }
 else
 {
   echo "Récapitulatif des informations saisies<BR>\n <UL>
    <LI>Nom: $nom</LI>
    <LI>Prenom: $prenom</LI>
    <LI>Age: $age</LI>
   </UL>";
 }
?>
```

## 7.9 Accès aux bases de données

PHP permet un interfaçage très simple avec un grand nombre de bases de données. Lorsqu'une base de données n'est pas directement supportée par PHP, il est possible d'utiliser un driver ODBC, pilote standard pour communiquer avec une base.

```
//connection :
$connect = mysql_connect($host,$user,$password)
               or die("erreur de connexion au serveur $host");

//$user : Le nom d'utilisateur
//$passwd : Le mot de passe
//$host : L'hôte (ordinateur sur lequel le SGBD est installé)

//choix du SGBD :
mysql_select_db($bdd) or die("erreur de connexion a la base de donnees");

//$bdd : Le nom de la base de données
```

Il est possible d'invoquer la clause (fonction) die() en cas d'erreur d'exécution. Si la fonction retourne la valeur 0 (c'est-à-dire s'il y a une erreur) la fonction die() renvoie un message d'erreur.

Lorsque l'on effectue une requête de sélection de tuples à l'aide de la fonction `mysql_query`, il est essentiel de stocker le résultat de la requête (les enregistrements) dans une variable, que l'on peut nommer par exemple $result.

Toutefois, cette variable contient l'ensemble des enregistrements et n'est donc pas exploitable telle quelle. Aussi utilise t-on l'une des fonctions `mysql_fetch_row()` ou `mysql_fetch_array()`, qui découpe les lignes de résultat en colonnes (par exemple Nom,adresse,...) et les affecte à une variable tableau dans l'ordre où elles arrivent.

Ainsi, imaginons une table appelée liens contenant le nom et l'URL de sites internet. Il est possible de récupérer l'ensemble des enregistrements et de les afficher dans un tableau :

```
<html>
<head>
<title>Liens</title>
<head>

<body>

 <table border="1" cellpadding="0" cellspacing="0">
 <tr>
  <th>Nom du site</th>
  <th>URL</th>
  </tr>


<?php
 // Déclaration des paramètres de connexion
$host = la_machine;
 // Généralement la machine est localhost
 // c'est-a-dire la machine sur laquelle le script est hébergé

 $user = votre_login;
 $bdd = Nom_de_la_base_de_donnees;
 $password  = Mot_de_passe;

 // Connexion au serveur
 mysql_connect($host, $user,$password) or die("erreur de connexion au serveur");
 mysql_select_db($bdd) or die("erreur de connexion a la base de donnees");

 // Creation et envoi de la requete
 $query = "SELECT nom,prenom FROM personnes ORDER BY nom";
 $result = mysql_query($query);

 // Recuperation des resultats
 while($row = mysql_fetch_row($result))
 {
  $nom = $row[0];
  $prenom = $row[1];
   echo "<tr>\n
```

```
  <td><a href=\"$prenom\">$nom</a></td>\n
  <td>$Url</td>\n
  </tr>\n";
 }
 // Deconnexion de la base de donnees
 mysql_close();
?>

</tr>
</table>
</body>
</html>
```

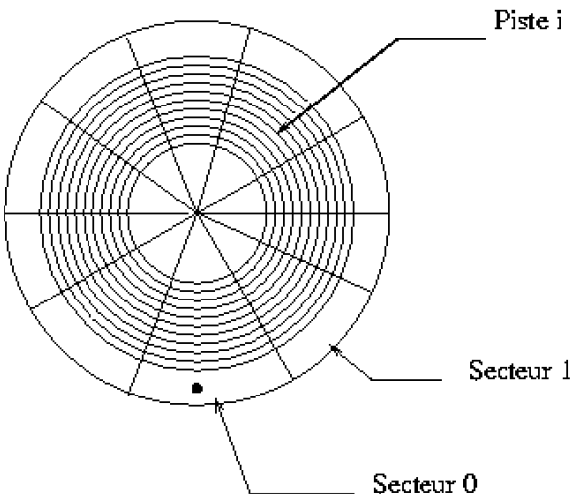
Dans l'exemple ci-dessus, les requête retourne les champs `nom` et `prenom`. La fonction `mysql_fetch_row()` analyse donc chaque ligne de résultat de la requête et stocke les colonnes dans le tableau `row[]`. Ainsi, le champ nom sera stocké dans row[0] et prenom dans row[1]. D'autre part, on inclue généralement `mysql_fetch_row()` dans une boucle TantQue de telle façon à ce que l'ensemble des lignes de résultat soient traitées. Lorsqu'il n'y a plus de ligne à traiter, la boucle TantQue se termine et l'interpréteur exécute la suite des instructions.

La non présence de résultat se traduit par un résultat nul de la part de la fonction `mysql_fetch_row()`.

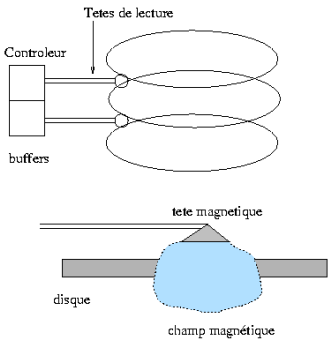
# Architecture des SGBD

## 1 Structure des disques

Un disque est composé d'une série de galettes magnétiques tournant autour du même axe. Les faces intérieures, magnétisées, sont physiquement constituées d'une série de pistes numérotées, et logiquement découpées en secteurs eux aussi numérotés. Un trou laissant passer la lumière d'une diode permet de connaître le numéro du secteur situé en face des têtes de lecture à un instant donné.



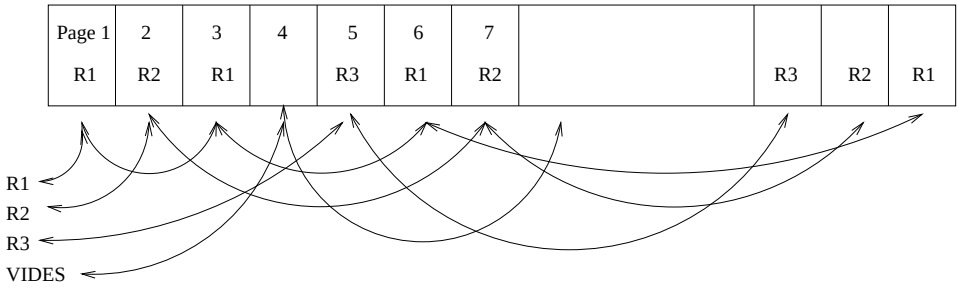

La nécessité de diminuer la taille des pistes fait que le champ magnétique permettant la lecture/écriture doit être aussi réduit que possible. De ce fait, les têtes de lecture doivent flotter au plus près de la surface, ce qui pose des problèmes d'intégrité des données et, en corollaire, de temps d'accès aux données. Le délai moyen d'attente (temps d'inertie) lors d'une opération d'E/S et de l'ordre de 4 à 5 ms. De ce fait, les E/S sont bufférisées et concernent, non un octet, ni même un secteur, mais la portion de cylindre constituée par toutes les fractions de pistes situées dans le même secteur à la verticales les unes des autres.



# 2 Organisation des fichiers

## 2.1 Pagination

Les fichiers des SGBD sont souvent des fichiers paginés, du moins dans les grands produits commerciaux (Oracle). La dimension d'une page correspond en principe à celle d'un cylindre de façon à pouvoir être accessible en une seule opération d'Entrée/Sortie. Au sein du fichier, les pages sont gèrées comme une liste chaînée. Les principaux éléments (tables, index) occupent plusieurs pages, réparties au sein du fichier au gré des mécanismes d'allocations. Chaque page est alors liée à la suivante, qui n'est pas nécessairement contigue, et qui contient la suite des données de l'objet concerné. Les pages libres, gèrées par une liste particulière, sont elles aussi chaînées, de façon à pouvoir être attribuées lors de toute demande d'allocation.




*Structure du fichier paginé des relations*

## 2.2 Clustérisation

C'est un procédé qui permet d'optimiser le traitement de certaines jointures, en stockant physiquement les éléments les plus fréquemment demandés au sein d'une même page, quelle que soit la relation à laquelle ils appartiennent. Les clusters sont créés par l'ordre `SQL : Create Cluster`. Ils sont référencés au moment de la création d'une table, afin de permettre un stockage sélectif des attributs. La clustérisation nécessite des moyens d'accès spécifiques aux divers attributs de la même table.

exemple :

```
create cluster C(A number,B char(10), D number);
create table T1(A number, B char(10), C date) cluster C(A,B);
create table T2(D number, E char) cluster C(D);
```

## 2.3 Bufférisation

Dans tout système, les entrées/sorties sont bufférisées et un SGBD ne déroge pas à la règle. Dans un système informatique classique, on utilise la stratégie `LRU` ( *last recently used*) : Les buffers sont chargés au fur et à mesure des besoins de lecture. Lorsqu'ils sont saturés, le contenu du plus ancien buffer est écrasé au profit des données réclamées.

Dans un SGBD, l'exécution d'une jointure, qui réclame en général beaucoup de buffers, demande l'écrasement du buffer le plus récemment chargé : c'est la stratégie `MRU` (*most recently used*).

En fait, la gestion d'un SGBD réclame des mécanismes plus sophistiqués que celle d'un système classique. En particulier, la sauvegarde des données lors des écritures est régie par des protocoles qui garantissent, comme nous le verrons plus loin, la sécurisation des transactions.

# 3 Interface avec la couche logique

L'utilisateur d'une BD relationnelle ne manipule que des tables, qui correspondent au niveau logique. Le niveau physique est constitué d'un ou de plusieurs fichiers paginés dont la structure peut être assez complexe. Une interface doit mettre en rapport ces deux niveaux. Cette dernière est un descripteur des zones physiques et doit contenir les têtes des listes chaînées associées à chaque relation.

Cette interface est assurée par exemple dans `ORACLE` par la notion de `Tablespace` qui est manipulé par les instructions `SQL create,alter,delete Tablespace`. Des paramètres physiques (clause `Storage`) permettent de préciser la dimension des blocs, ainsi que leur mode d'extension lors des allocations.

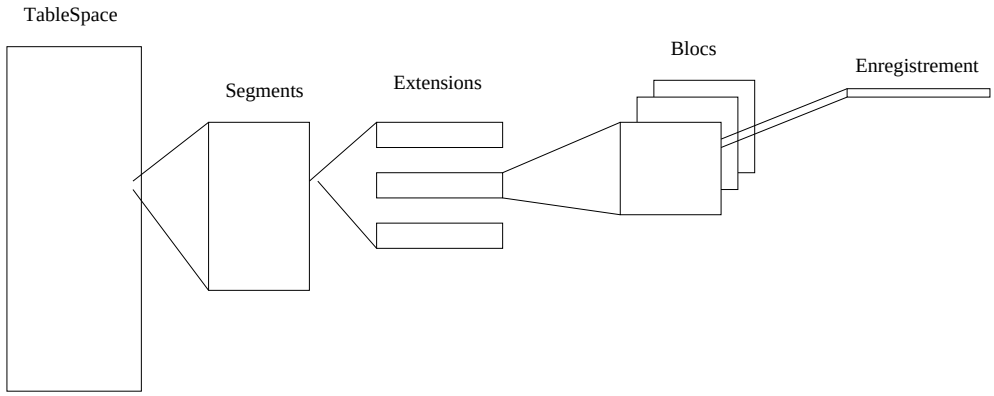
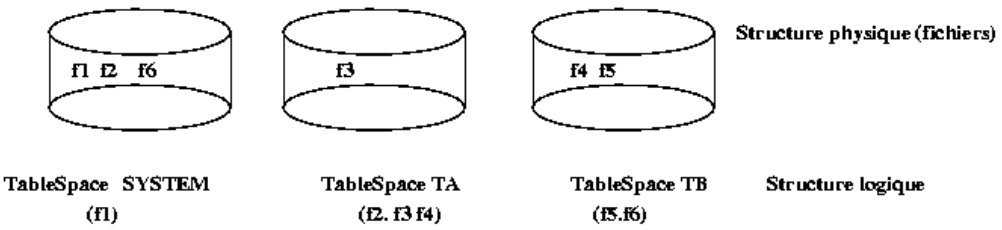



*Schéma d'organisation d'un TableSpace et de l'accès aux données*

## 3.1 Création d'une base (Oracle)

La création d'une base se fait par l'ordre `create database`, qui permet d'associer la base à un certain nombre de fichiers constitués d'une série contigüe d'adresses sur le disque.

```
CREATE DATABASE maBase logfile 'journal.log', size 50MB, datafile 'fichier1.ora' size 2GB;
```




Lors de la création d'un utilisateur, ce dernier est rattaché à un tablespace, SYSTEM par défaut.

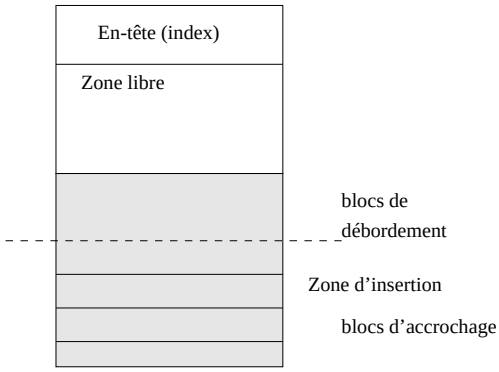
```
CREATE USER toto IDENTIFIED BY truc DEFAULT TABLESPACE TA,
  TEMPORARY TABLESPACE TA, QUOTA UNLIMITED ON TA, QUOTA 10MB ON SYSTEM;
```

De même, lors de la création d'une table, on peut l'attacher à un tablespace particulier si on le souhaite :
`CREATE TABLE T(...) TABLESPACE TB;`

## 3.2 Stockage des tuples (Oracle)

Chaque bloc ou page de fichiers est muni d'un index permettant de gérer les lignes avec efficacité. A chaque enregistrement est associé un bit de validité permettant une destruction logique et non physique, qui serait bien trop coûteuse.

Lorsque les enregistrements sont de longueur variable, la page contient une partie fixe, le *bloc d'accrochage*, et un pointeur permettant d'accéder à la partie variable (le *bloc de débordement*) qui est éventuellement stocké dans une autre page du fichier. Un exemple d'enregistrement de taille variable est constitué par le type `Varchar2` ou le type `long` qui peut atteindre 65000 caractères.

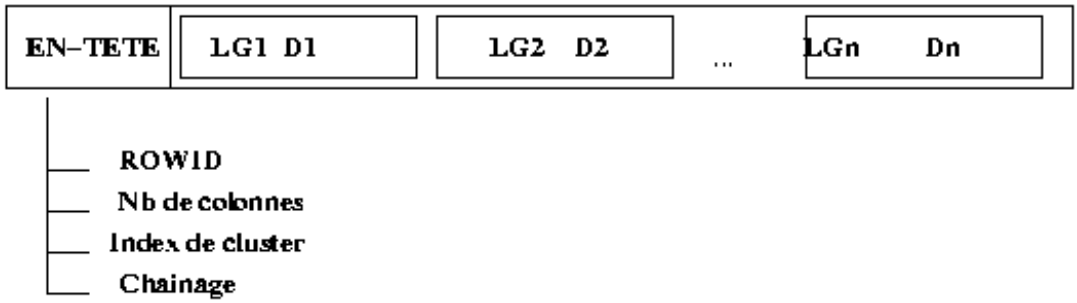



*Structure d'un bloc de donnée*

Lors de la création d'une table, une clause **storage** indique le volume initial de l'allocation (**initial**), le volume alloué lors d'une demande ultérieure (**next**), le nombre d'extensions d'allocation qu'il est possible de demander, le pourcentage d'augmentation du volume à chaque demande ultérieure (**pctincrease**), ainsi que le numéro de la liste permetant de gérer l'espace libre en cas d'insertion/suppression de tuples. En voici un exemple :

```
Storage ( initial 15 KB, next 10KB, minextents 3, maxextents 15,
                                        pctincrease 20, freelist 3);
```

Au sein d'un bloc, la structure des enregistrements est la suivante :




Dans ce schéma, LGi désigne la longueur de la donnée Di qui suit, et qui correspond à la représentation binaire du ième attribut.

## 3.3 Dictionnaire des données

La *métabase* est la base qui gère toutes les bases. Elle est conçue selon un schéma relationnel et constitue un dictionnaire des données prenant en compte :

- les utilisateurs
- leurs droits
- les objets (tables, vues, index, etc)
- les attributs de chaque table
- la structure logique de description des tables, la taille et l'organisation des blocs de données.
- l'organisation des fichiers, leur taux d'occupation, l'espace libre.

A titre d'exemple, le dictionnaire d'ORACLE était constitué de 44 tables pour la version 5, d'une centaine pour la version 6, près de 300 pour la version 7 et plus de 1000 pour la V10. Certaines de ces tables ne peuvent être accédées que par le DBA.

La base du dictionnaire de la V7 est en réalité détenue par le super-administrateur `SYS`, qui voit les tables suivantes (toutes ne sont pas indiquées) du véritable dictionnaire :

| | |
|---|---|
| **USER$** | Liste des utilisateurs et des rôles |
| **OBJ$** | Objets des utilisateurs |
| **TS$** | Tablespaces |
| **FILE$** | Fichiers |
| **SEG$** | Segments |
| **TAB$** | Tables de la base |
| **CLU$** | Clusters |
| **IND$** | Indexes |
| **ICOL$** | Colonnes accessibles par un index |
| **CON$** | Liste des contraintes |
| **CDEF$** | Définitions des contraintes |
| **CCOL$** | Colonnes sur lesquelles portent des contraintes |

Le dictionnaire courant accessible à l'administrateur `SYSTEM` ainsi qu'aux utilisateurs ordinaires est composé de vues spécialisées, permettant d'avoir la liste des objets de la base. Ce sont des vues construite sur le dictionnaire interne, même quand elles sont qualifiées de tables. Certaines sont issues des anciennes versions d'`Oracle` pour des raisons de compatiblité.

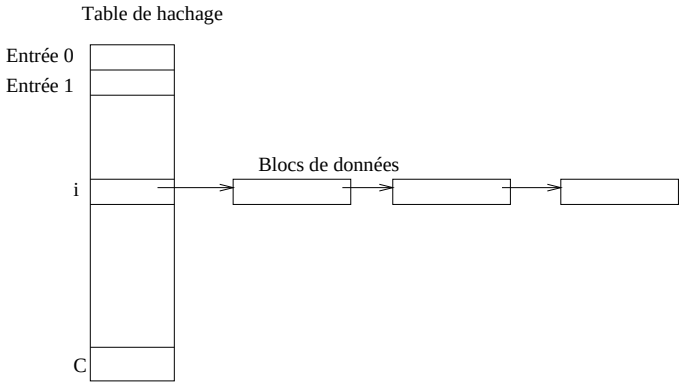
# 4 Les méthodes d'accès

Aujourd'hui, les SGBD utilisent essentiellement des accès directs en utilisant les techniques de hachage et d'indexation.

## 4.1 Le hachage

### 4.1.1 Principe

Le principe en est très simple. L'accès est calculé par une *fonction de hachage* h, qui, à une entrée i, fait correspondre la valeur h(i) $\in [1, \mathbb{C}]$. Une même valeur h(i) peut correspondre à plusieurs entrées différentes, qui appartiennent alors à la même classe d'équivalence. La fonction h est caractérisée par la valeur $\mathbb{C}$ du nombre de classes qu'elle détermine.




L'accès aux données est réalisé par une *table de hachage*, qui, à chaque valeur de h(i), fournit l'adresse de la zone de stockage correspondante. Comme plusieurs entrées délivrent la même valeur, on assiste, relativement à la zone de stockage, à un mécanisme de *collision* qui peut être résolu de différentes façons ; les plus courantes sont le chaînage, ou l'utilisation d'une *fonction de collision* pour obtenir une nouvelle adresse. Dans tous les cas, la recherche de l'information devient séquentielle, chaque collision imposant la lecture ou le calcul de l'adresse suivante.

### 4.1.2 Exemples

Il n'y a pas de théorie pour construire une "bonne" fonction de hachage, chaque programmeur conservant jalousement celle qu'il a réussi à imaginer après un bricolage plus ou moins laborieux.

Le principe du hachage suppose une équi-répartition des valeurs de l'argument, ce qui est rarement observé en pratique. Cet argument est toujours numérique, et necessite une transformation préalable des clés alpha-numériques.

Une bonne méthode de transformation des clés alph-numériques en valeurs numériques consiste à extraire des caractères médians d'une clé ( 2 ou 4), et à prendre les valeurs binaires correspondantes pour former les poids

hauts et bas d'une entier. Une fois la clé numérisée, il existe bien des procédés pour construire des fonctions de hachage :

- Les fonctions les plus simples sont construites à partir de la fonction *modulo* : h(clé)=clé Modulo N, ou N est le nombre maximum de classes voulues.
- La méthode du *milieu des carrés* consiste à extraire des bits du milieu du carré des clés. La valeur obtenue correspond à une classe d'équivalence et est convertie en adresse. Exemple : $5486^2 \rightarrow 30096196$. On peut extraire la valeur médiane : 96.
- la méthode de *découpage et addition* transforme un nombre sans tenir compte de la dernière retenue, conformément à l'exemple suivant : $24315268 \rightarrow 243 + 152 + 68 = 463$

### 4.1.3 Accès multiclés

Les tables de hachage peuvent être bi, voire multi-dimensionnelles. Elles permettent en cas de recherche multicritères d'accéder à des zones où sont stockés les enregistrement correspondant à des entrées multiples. La gestion d'une telle structure est cependant très lourde si l'on tient compte des mises à jour.

On peut aussi réaliser un hachage partitionné si la fonction de hachage est capable de traiter une entrée constituée de champs indépendants. Par exemple, si le premier champ renvoit à un service et le dernier à un emploi, la valeur 010.110 permet d'accèder à un programmeur particulier d'IEEA. La valeur 000.110 renverra à l'ensemble des programmeurs.

### 4.1.4 Tableaux de bits

Une utilisation annexe, mais importante, des fonctions de hachage est liée à l'optimisation des opérations de jointure. De façon à ne sélectionner que les tuples qui interviennent dans la jointure de R et S, lorsque la cardinalité de R est très grande devant celle de S, on opère de la façon suivante :

On applique aux tuples de S une petite série de fonctions simples, $h_0$, $h_1$, ..., $h_n$, ayant si possible le même nombre de classes. Si, pour un tuple $s_i$ de S, h($s_i$)=j, on met à 1 l'élément (i,j) d'une table T, et à 0 sinon. Cette table sert ensuite de crible pour tester les tuples de R. Si un tuple de R a une signature conforme à la table ($\forall$i, $h_i(r) = j$ avec T(i,j)=1), il est retenu, car susceptible de participer à la jointure.

| Classes | 012345 ..... j ..... N |
|---|---|
| h0 | 10110101010111010100010101 0 |
| h1 | 01010101111010100010101010 1 |
| h2 | 11100110001010001111010101 1 |
| h3 | 00110000110101110101110010 1 |

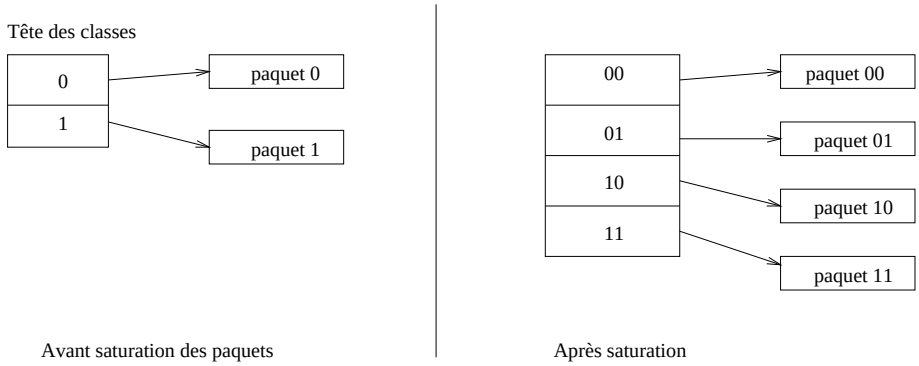
*Tableau de bits ou crible de jointure*

### 4.1.5 Gestion des collisions

C'est le principal problème à résoudre si on veut éviter la séquentialisation qui en découle. En cas de collision, il faut ranger un tuple dans l'espace associé à une classe. Si cet espace est saturé, il faut allouer une extension de taille suffisant e pour tenir compte d'autres collisions éventuelles.

La méthode de Fagin conduit à une croissance quadratique de l'espace d'allocation. Son principe est le suivant :

Lors de l'initialisation d'une table, on utilise une fonction de hachage dont la valeur tient sur un bit. Elle permet d'accéder à seulement 2 paquets dénommés 0 et 1. Dès qu'un des paquets est saturé, on alloue le même nombre de paquets, ce qui fait 4 au total. La fonction de hachage, compte tenu de la collision qui vient de se produire est alors sur 2 bits, et les valeurs dans les paquets initiaux sont réparties en fonction de la valeur de h(clé, c). Le paramètre c, qui indique le nombre de collisions, vaut alors 1. le procédé est réitéré ensuite autant de fois que nécessaire, mais la saturation de l'espace mémoire peut intervenir à tout moment.




*Méthode de Fagin*

La méthode de Litvin permet d'obtenir une croissance linéaire de l'espace mémoire. Elle utilise une fonction de hachage et une fonction de collision. Supposons par exemple que $h_0$ = clé modulo N, avec N=10. $h_0$ fournit donc les adresses de 10 blocs différents.

Soit la fonction de collision $h_{c+1} = h_c + \mathrm{N}\times 2^c$. Lors de la première collision, on crée le nombre de blocs nécessaires pour correspondre à la valeur de $h_{c+1}$. Si 26 est une clé, $h_0(26)$ vaut 6, et le $6^{eme}$ bloc est saturé. $h_1(26)$ vaut 16. On crée les blocs de 10 à 16, et on redistribue certaines des valeurs des blocs de 0 à 9 vers les blocs de 10 à 16, pour permettre des insertions directes vers les premiers blocs.


Les 2 dernières valeurs des blocs 0 et 6 sont réparties dans les blocs vides 10 et 16, respectivement.
La valeur 26 est alors insérée dans le bloc 6.

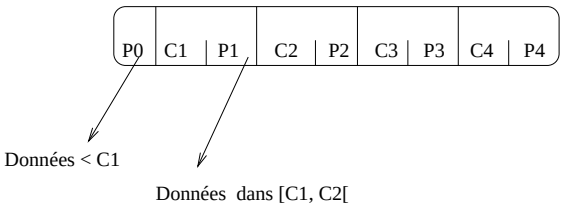
*Méthode de Litvin à croissance linéaire*

## 4.2 Index arborescents

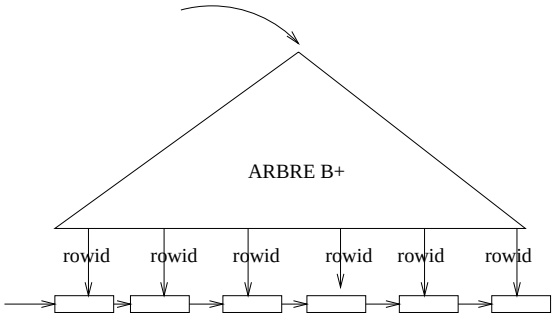
Les index arborescents sont réalisés à partir d'arbres équilibrés de la famille des B-arbres, principalement les arbres $B^+$.

Un arbre $B^+$ d'ordre d est un arbre totalement équilibré (toutes ses feuilles sont à la même hauteur) dont les nœuds sont composés d'un nombre de valeurs comprises entre d+1 et 2d+1 (sauf éventuellement la racine).

*Structure du noeud d'un B-Arbre de degré 2*




La première valeur d'un nœud est une adresse (un pointeur), les autres sont des index : (clé, adresse). L'adresse correspond à un autre nœud, de hauteur inférieure, ou à une feuille. En ce qui concerne les feuilles, l'adresse est celle d'une page contenant le tuple recherché, et, sous Oracle, correspond au `rowId`. Pour les nœuds, l'adresse associée à $C_i$ renvoit au nœud contenant les références des clés comprises entre $C_i$ et $C_j$, $C_j$ *exclu*.




La nombre de niveaux nécessaire pour stocker N clés est $1 + log_{d/2}\frac{(N+1)}{2}$ et le coût d'un accès est en $log_d\ N$. A titre d'exemple, 3 accès suffisent pour $10^6$ clés lorsque $d = 200$.

La principale difficulté de gestion d'un B-arbre est de conserver les feuilles au même niveau lors des insertions et des suppressions. Rappelons les algorithmes exposés dans le cours d'algorithmique.

### 4.2.1 Recherche

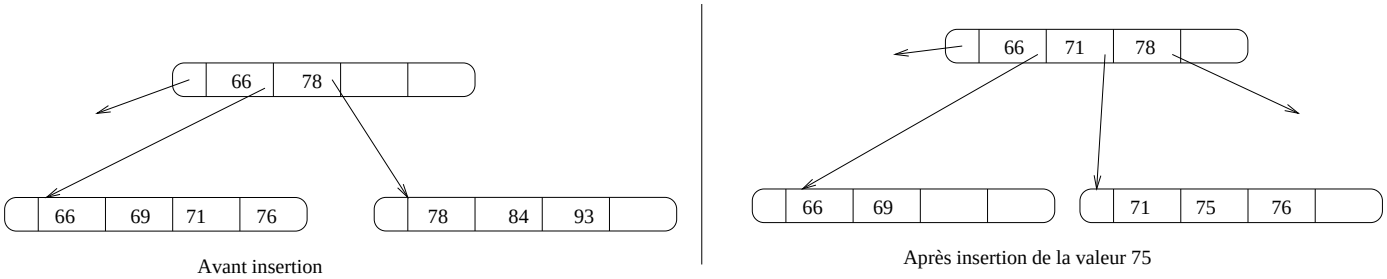
L'exploration du contenu d'un nœud se fait séquentiellement dans l'ordre des champs, les clés étant stockées dans l'ordre alpha-numérique.

Lorsque la clé cherchée n'est pas présente, la recherche se poursuit au niveau du nœud pointé par l'index dont la clé est immédiatement inférieure. S'il n'en existe pas, c'est le *pointeur antérieur* $P_0$ qui indique le nœud suivant.

### 4.2.2 Insertion

On commence par effectuer une recherche du nœud où doit se faire l'insertion. S'il n'est pas saturé, l'insertion se fait en réorganisant les index présents de façon à respecter l'ordre alpha-numérique.

Si le nœud est plein, il faut créer un nouveau nœud, en éclatant celui qui est saturé. Les index correspondant aux d premières clés (les plus petites) restent sur place, les d plus grandes sont insérées dans le nouveau nœud, et la valeur médiane est remontée au niveau du nœud père. Le processus est répété récursivement si nécessaire. En cas de besoin, on crée un nouveau nœud racine, ce qui permet à l'arbre par le haut, les feuilles *restant toujours au même niveau.*

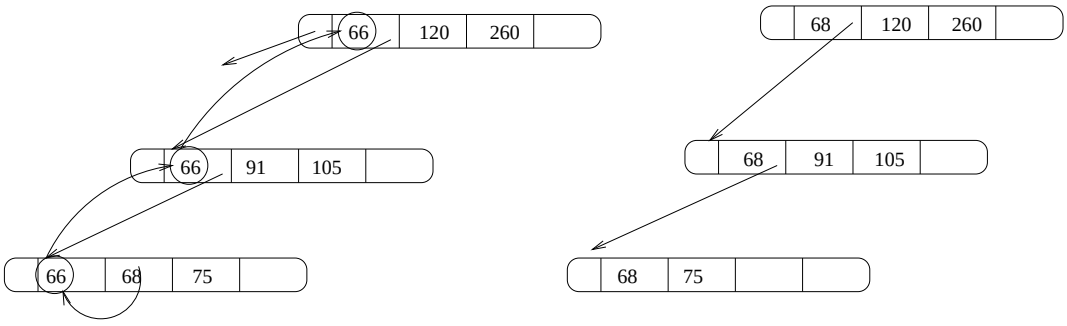



*Insertion de la valeur 75 dans un arbre B+*

### 4.2.3 Suppression

L'algorithme dépend de la position de la valeur à supprimer :

- Si le nombre d'index est supérieur à d, on supprime la valeur du nœud que l'on réorganise.
- Si la valeur est située dans une feuille et que le nombre d'index qu'elle contient est égal à d, on recombine les feuilles voisines.
- Si l'index est dans un nœud non feuille, on fait la suppression dans la feuille associée et on réorganise récursivement les nœuds supérieurs. Il est possible ainsi de supprimer la racine ; ce qui a poour conséquence l'adoption d'une nouvelle racine au niveau inférieur et la diminution d'un niveau pour l'arbre.




Suppression de la clé 66

# Sécurité des données

## 1 Résistance aux pannes

### 1.1 Mémoire sûre

Toutes sortes de pannes peuvent survenir lors de l'exécution d'une transaction :
- des erreurs logicielles, bugs, oubli du commit, mauvaise analyse du traitement, ...
- des pannes du serveur ou de la machine client
- des pannes de réseau (coupure, saturation, ...)
- des pannes de disque (crash, panne du contrôleur).

Le SGBD doit garantir la cohérence de la base et l'atomicité des transactions. Il doit donc assurer différents niveaux de protection, de façon à garantir la sauvegarde des données quel que soit l'environnement de traitement. Cette préservation est assurée par les protocoles d'écriture basés sur la notion de *mémoire sûre* mis en œuvre par le moyen de la *journalisation* des transactions.

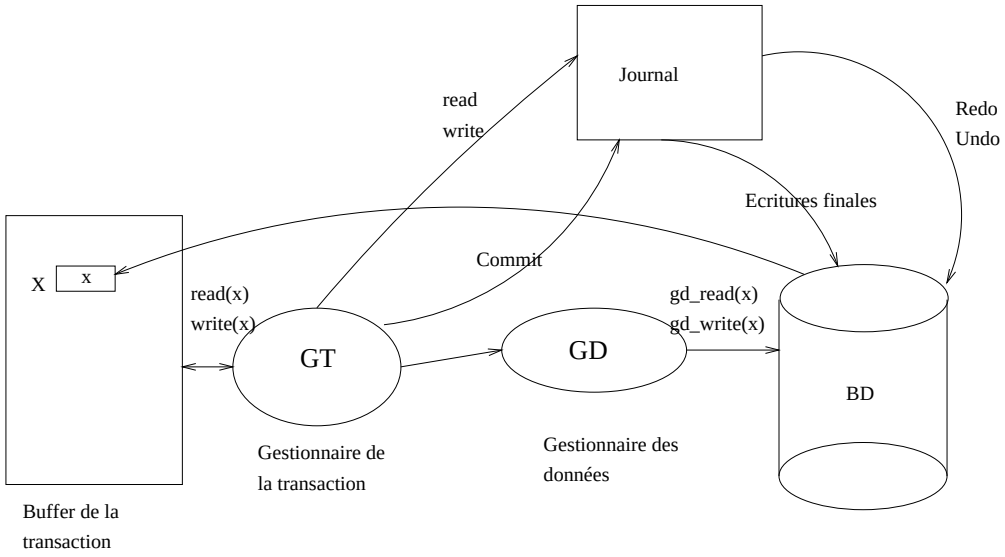
### 1.2 Journalisation

Toutes les opérations d'accès à la base sont enregistrées dans plusieurs journeaux, selon un principe d'écritures en miroir. Dans un cadre professionnel, il est préférable que les différentes versions du journal soient stockées dans des disques différents abrités dans des sites différents, de façon à se mettre à l'abri des accidents, incendies, sabotages, ...Il est bon également de disposer d'une sauvegarde sur des supports non magnétiques (papier, cdrom).

Les différentes opérations de la vie des transactions enregistrées dans le journal sont entrelacées du fait de la concurrence. Elles sont donc distinguées par un identificateur de transaction qui peut être une estampille. On peut ainsi trouver, pour une transaction :

```
<Ti, debut de transaction>
<Ti, lecture, variable, valeur>
<Ti, ecriture, variable, nouvelle valeur, ancienne valeur>
<Ti, commit>
<Ti, rollback>
<Ti, point de controle>
<Ti, fin de transaction>
```

L'atomicité de la transaction est prise en compte de la façon suivante : Au moment ou est atteint l'état de validation partielle ou logique, le gestionnaire de la transaction assure la sauvegarde des opérations en mémoire sûre. Les écritures dans le journal sont assurées par un processus particulier qui envoit un acquittement lorsque tout est enregistré. Le gestionnaire de transaction effectue alors la sauvegarde effective des données (`commit`) en les transmettant au gestionnaire des données qui réalise les écritures sur les disques de la base.




Lorsqu'une panne survient au cours du déroulement d'une transaction, elle peut intervenir avant ou après son point de validation totale. Les journaux sont alors lus. Si le commit est enregistré dans le journal, c'est que la transaction s'est achevé normalement. Il suffit de reprendre l'ensemble des opérations enregistrées (processus `redo`) pour restaurer les données. Si ce même processus de récupération tombe en panne, l'opération peut-être reprise autant de fois que nécessaire : elle est *idempotente*

Par contre, si le commit est absent du journal, il faut restaurer les données dans leur état antérieur à la transaction (processus `undo`). Là encore, il suffit de reprendre le journal autant de fois que nécessaire, ce processus étant lui aussi idempotent. La plupart du temps, il n'est pas nécessaire d'effectuer la lecture de la totalité du journal qui est souvent volumineux. Les opérations `undo` et `redo` peuvent parfaitement être menées à partir d'un *point de contrôle* que le programme met en place entre deux phases indépendantes du traitement.

# 2 Sécurité

## 2.1 droits et rôles

La sécurité est assurée à plusieurs niveaux. La connexion à la base peut-être contrôlée par un mot de passe. `SQL` permet de plus de définir différents niveaux d'accès.

Les droits sont assurés globalement et pour chaque objet de la base :

```
grant connect|resource|DBA to uid identified by pswd
```

Le mot-clé `connect` ne donne que des droits de lecture, `resource` permet de créer et de modifier ses objets, et `DBA` correspond aux droits de l'administrateur qui peut tout (ou presque) se permettre.

Les manipulations d'objet sont assurées par :

```
grant select|insert|update|delete|index|cluster|alter|all on objet to utilisateur
                                [with grant option]
```

Ces droits sont donnés par le propriétaire à un usager, avec éventuellement la possibilité pour ce dernier de le transmettre à son tour (clause `with grant option`).

on peut trouver fastidieux de délivrer des droits identiques à des utilisateurs différents. Il paraît plus simple de définir des types de droits que l'on affecte ensuite aux bénéficiaires : c'est le concept de rôle. On peut ainsi trouver la séquence suivante (Oracle) :

```
create role R;
grant select, update on EMP to R;
grant R to toto;
revoke R from toto;
drop role R;
```

## 2.2 Protection des données

### 2.2.1 Utilisation de vues

La vue est un moyen efficace de protection des données et de restriction des accès. Elle permet de limiter la vision des données au niveau des lignes et des colonnes de la seule vue. En règle générale, un service informatique qui a le souci de protéger ses données sensibles ne doit jamais donner à un utilisateur l'accès directement aux

tables, mais n'utiliser à cet effet que des vues. L'administrateur doit cependant être attentif aux limitations de mises à jour à travers une vue qui peuvent rendre impossible certains types de traitements.

## 2.3 Procédures stockées

Une autre possibilité d'assurer une sécurité optimale est dans ce cas l'utilisation obligée de procédures stockées (PL/SQL). Celles-ci, propriété de l'administrateur, bénéficient de ses droits, mais ne permettent qu'un traitement spécifique. L'utilisateur disposant d'un droit d'exécution pourra agir sur la base selon le mode autorisé sans même avoir à en connaître la moindre caractéristique (nom des tables et des vues manipulées).

### 2.3.1 Niveaux de sécurité

Il existe plusieurs normes (NCSC aux USA, ITSEC en europe). Celles-ci définissent des niveaux de sécurité en ce qui concerne les données et les programmes d'application (ou les utilisateurs) ; par exemple, pour NCSC, par niveau décroissant :

- top-secret (ts)
- secret (s)
- confidential (c)
- unclassified (u)

La lecture des données peut se faire pour des données de niveau inférieur ou égal à celui du programme ou de l'utilisateur. L'écriture ne peut concerner que les données de même niveau. Ce dernier est précisé dans les tables par un attribut supplémentaire.

### 2.3.2 Cryptage

Ce dernier reste en France limité par la loi, bien qu'on trouve des cartes de cryptage à un prix modique, et des algorithmes puissants (PGP) gratuitement sur Internet. Il est soumis à autorisation auprès du ministère des armées et ne concerne que certains services d'état ainsi que des banques pour certaines opérations confidentielles. Le cryptage est cependant admis pour la transmission de mots de passe et pour les signatures électroniques. L'usage du "tiers de confiance" est aussi inscrit dans la loi. Une société peut crypter ses données à condition de confier la clé du cryptage à un organisme de confiance agréé par le ministère des armées.

### 2.3.3 La recherche par inférence

Elle peut être puissamment réalisée à l'aide d'outils conçus pour les services de renseignement militaires, aujourd'hui largement utilisés dans le civil. Ces logiciels assurent des fonctions d'espionnage économique contre lequel certaines entreprises doivent se protéger.

Parmi ces outils, citons *Topic*, créé par la CIA, *L4U/Taïga* par le DGSE, *Spirit* développé par le CEA, *Dataview* du centre de recherches rétrospectives de Marseilles, *Périclès* conçu pour la marine, *Technology Watch* d'IBM. Certains font de la navigation automatique dans Internet, d'autres consultent des bases de données à la recherche d'informations qu'ils recoupent et synthétisent.

Cette recherche d'information (*Data Mining*) commence (1996) à constituer une activité spécifique en informatique et touche à tous les domaines (banques, finance, industrie, recherche, santé,...)