# LES SYSTEMES DE GESTION DE BASES DE DONNEES

## VERSION 2.0

## MANUEL DE L'ELEVE

**Pierre Stockreiser**
**Lycée Technique d'Esch-s-Alzette**
**Août-Septembre 1999**

Je tiens à remercier M. Sylvain Piren, Professeur-Ingénieur au Lycée Technique d'Esch-s-Alzette, pour avoir essentiellement contribué à la rédaction de ce cours.

Je remercie également les personnes suivantes pour leur support respectivement leur influence pendant le travail de recherche et de rédaction.

M. Christian Lucius, Professeur de Sciences au Lycée Technique Michel-Lucius
M. René Weber, Professeur-Ingénieur au Lycée Technique des Arts et Métiers
M. Jean-Marie Ottelé, Professeur-Ingénieur au Lycée Technique Ecole de Commerce et Gestion

P. Stockreiser

# Préface

Ce document est un support pour les cours en informatique des classes de 13CG. La structure et le contenu des chapitres de ce document ont été synchronisés avec le contenu du programme établi par la CNPI. Le cours met l'accent sur les concepts et techniques fondamentaux des bases de données relationnelles, ainsi que sur la conception et l'implémentation de systèmes informatiques élémentaires de gestion.

Le cours est subdivisé en quatre parties:

| | |
|---|---|
| PARTIE 1 : Modélisation d'un système d'information | (chapitres 1 – 4) |
| PARTIE 2 : Exploitation des bases de données relationnelles | (chapitres 5 – 9) |
| PARTIE 3 : Protection des données | (chapitre 10) |
| PARTIE 4 : Travaux sur logiciel | (chapitre 11) |

Ce cours n'est pas du tout un manuel d'utilisation de MS-Access, de Win'Design respectivement d'un autre logiciel. Le cours se limite aux concepts importants en relation avec le sujet. Dans les rubriques TP, seulement les manipulations les plus importantes sont mentionnées.

**Symboles utilisés à l'intérieur de cet ouvrage:**

| | |
|---|---|
|  | **Paragraphe important** |
|  | **Exercice** |
|  | **Exercice TP** |
|  | **Remarque pédagogique** |
|  | **Manipulation sur logiciel** |

# Table des matières:

# Partie 1 : Modélisation d'un système d'information

# 1. Analyse des systèmes d'information

## 1.1 Introduction

La compétitivité d'une entreprise ainsi que sa valeur sur le marché sont déterminées par plusieurs éléments, d'une importance différente selon le secteur d'activité. On peut généralement regrouper ces éléments en deux classes:

1. Les éléments matériels
    - L'infrastructure
    - Les supports financiers

2. Les éléments intellectuels
    - La compétence des employés
    - La motivation des employés
    - **Le recueil et l'exploitation optimale des informations utiles**

Depuis quelques années, les responsables des entreprises (banques, assurances, industrie etc. ) ont davantage reconnu et admis que la gestion et l'exploitation des informations sont un facteur de compétitivité à ne pas négliger.

Le développement rapide de l'informatique a donné aux entreprises la possibilité d'utiliser des moyens avancés et puissants pour gérer et exploiter de très grands volumes de données. Il y a quelques années, le domaine de la gestion informatique des données était réservé aux informaticiens. Actuellement, les tendances à l'intérieur des entreprises ont changé de façon à ce que tous les employés soient de plus en plus impliqués dans les différents procédés liés à la gestion et l'exploitation des données. De cette façon, **un certain niveau de connaissance des principes et des outils standards de l'informatique est aujourd'hui requis pour la plupart des postes disponibles dans les entreprises.**

Toutefois, il ne suffit pas d'utiliser les ressources informatiques les plus sophistiquées pour exploiter au mieux les données. En parallèle avec les outils informatiques utiles pour gérer des données, tels que les ordinateurs de plus en plus puissants et les logiciels adaptés (SGBD, Tableur etc.), ont été développées des méthodes d'analyse et de conception de systèmes d'information. Ces méthodes nous offrent la possibilité d'analyser un système d'information naturel, tel que par exemple la gestion des livres d'une librairie ou la gestion des sinistres d'une compagnie d'assurances, de concevoir ensuite un modèle qui représente ce système et d'implémenter finalement un système informatique, basé sur ce modèle.

## 1.2 Définition de l'information et des systèmes d'information

 **Une information est un élément qui permet de compléter notre connaissance sur une personne, un objet, un événement … .**

Exemple: Le nom d'une personne est une information concernant cette personne.
La couleur d'une voiture est une information concernant cette voiture.
La date de la fête scolaire est une information concernant cet événement.

 **Un système d'information est constitué par l'ensemble des informations relatives à un domaine bien défini.**

Exemple: Toutes les informations relatives à la gestion d'une librairie constituent le système d'information de cette librairie. Ce système peut couvrir le simple stockage des livres, mais également la gestion des commandes, des ventes et même des clients.

Un système d'information ne doit pas nécessairement être informatisé. Bien que la plupart des systèmes actuels se basent sur la technologie de l'informatique, il existe encore des systèmes d'information où l'information est stockée, manipulée et communiquée à l'aide de moyens "traditionnels" tels que armoires, classeurs, calculatrices, fiches sur papier etc. .

 **Un système d'information existe indépendamment des techniques informatiques.**

Le système d'information ne doit pas être confondu avec le système informatique qui est constitué par les éléments suivants:

- Les ordinateurs
- Les programmes
- Les structures de données (Fichiers, Bases de données)

Dans ce chapitre nous allons découvrir une démarche d'informatisation, qui nous permet de modéliser un système d'information et de le représenter à l'aide d'un système informatique. Le but de cette démarche est de concevoir des systèmes stables et optimisés en termes de performance, de fiabilité et de convivialité.

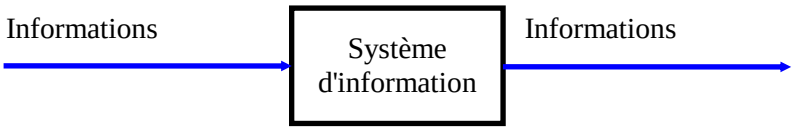
## 1.3 Les données, les traitements et les informations

Bien que les deux termes "informations" et "données" soient souvent utilisés comme synonymes[1], il existe une différence subtile entre eux.

Prenons un exemple:

Dans une librairie, un client demande au vendeur si le livre "L'étranger" (Albert Camus) est disponible en stock. Le vendeur conseille la base de données de la librairie à l'aide de son ordinateur et confirme au client que le livre est disponible. Le vendeur a donc donné au client **l'information** que le livre est en stock. Afin de pouvoir donner cette information, le vendeur a du consulter les **données** qui représentent le stock de la librairie. Le fait de consulter le stock constitue un **traitement** sur les données du stock.

Nous pouvons généraliser:




**Un système d'information contient les données et les traitements nécessaires pour assimiler et stocker les informations entrantes et produire les informations sortantes.**

Dans les systèmes d'information nous retrouvons généralement les traitements suivants:
- Consultation des données;
- Ajout de données;
- Suppression de données;
- Modification de données.

Exemple:

Le propriétaire d'une vidéothèque reçoit une livraison avec des nouvelles cassettes vidéo. Pour chaque cassette vidéo, il lit le titre, la langue et la durée et sauvegarde ces informations dans la base de données de la vidéothèque. Il a donc utilisé un traitement d'ajout de données afin de transformer les informations entrantes (titre, langue, durée) en données.

[1] Deux mots sont synonymes quand ils désignent une même chose.

## 1.4 La représentation informatique des données

Les données d'un système d'information peuvent être stockées et manipulées à l'aide d'un outil informatique spécialisé dans ce domaine. Actuellement les **Systèmes de Gestion de Bases de Données (SGBD)** constituent le type de logiciel le mieux adapté pour implémenter la plupart des systèmes d'information. Sachant que les tables forment la base de stockage d'une base de données, on peut représenter n'importe quel système d'information par un ensemble de tables dont chacune contient un certain nombre de champs de données. Nous allons voir qu'on peut même définir des liens entre ces tables via des champs communs.

# 2. Démarche de modélisation des données

## 2.1 Le groupe d'étude (angl. Project group)

Un système d'information qui n'est pas trop complexe et volumineux en terme d'informations, peut facilement être informatisé par une seule personne, qui ne doit pas nécessairement être un informaticien. Il suffit d'être un peu familiarisé avec une méthode de modélisation, et de savoir manipuler un SGBD pour réaliser une implémentation informatique, cohérente et fonctionnelle, d'un tel système d'information.

Dès que le système d'information atteint une certaine envergure (par exemple: informatiser la gestion des sinistres d'une compagnie d'assurances), un groupe d'étude est généralement créé.

Ce groupe ne devra en aucun cas contenir seulement des informaticiens mais également:

- Un ou plusieurs représentants des futurs utilisateurs du système informatisé
  (Par exemple: Un employé du service qui gère les sinistres) ;
- Un ou plusieurs représentants de chaque département impliqué
  (Par exemple: Un employé du service des contrats ) ;
- Un représentant de la direction.

Généralement, un responsable du groupe (angl. Project Manager) est nommé, afin de coordonner les travaux effectués par le groupe et de suivre le déroulement à partir de l'analyse jusqu'à la mise en place du système informatisé.

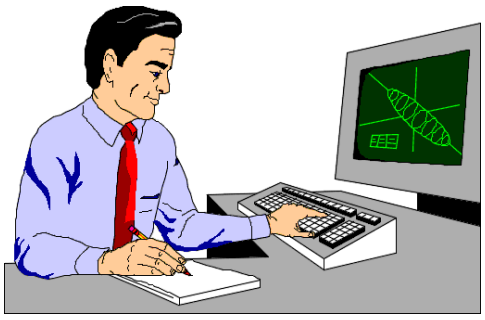
## 2.2 Les étapes

Chaque projet d'informatisation, qu'il soit exécuté par une seule personne, ou géré par un groupe d'étude, prévoit plusieurs étapes.

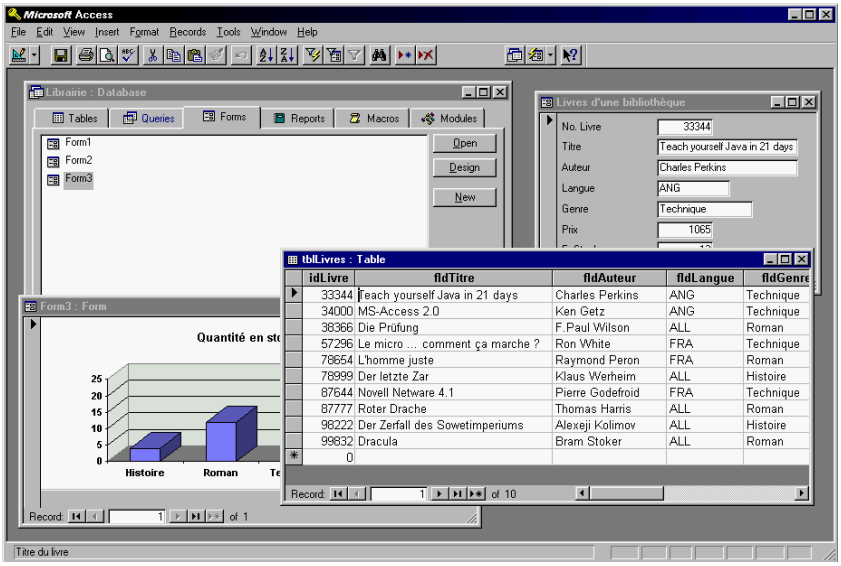
En général, nous avons les étapes suivantes:

1. Analyse de la situation existante et des besoins

2. Création d'une série de modèles qui permettent de représenter tous les aspects importants



3. A partir des modèles, implémentation d'une base de données



## 2.3 Sources d'information

La première étape de chaque projet est donc l'analyse de l'existant et des besoins. Afin de pouvoir réaliser une analyse correcte sur laquelle on peut baser la suite du projet, il faut d'abord identifier les sources d'information, et puis collectionner exactement les informations importantes pour le projet.

Sources d'information primaires:
- L'interview avec les utilisateurs;
- L'étude de documents provenant du système d'information actuel (Rapports, Bons de commandes, Factures …).

Pour les projets d'une certaine envergure s'ajoutent:
- L'interview avec les responsables des services impliqués;
- Pourvu que la tâche d'analyse soit partagée entre plusieurs membres du groupe d'études, il faut coordonner les actions et comparer les résultats avec les autres membres.

Pour les projets qui se basent sur un système déjà partiellement informatisé s'ajoute:
- L'étude de l'application informatique existante.

# 3. Méthode de modélisation des données

## 3.1 Définition

Nous avons vu que la démarche classique d'un projet informatique comprend les étapes suivantes:

1. Analyse de la situation existante et des besoins;
2. Création d'une série de modèles, qui permettent de représenter tous les aspects importants;
3. A partir des modèles, implémentation d'une base de données.

En ce qui concerne la première étape, nous n'allons pas introduire de vraies règles, mais simplement utiliser nos connaissances de gestion d'une entreprise, notre esprit ouvert et même notre fantaisie pour analyser correctement la situation existante et les besoins des utilisateurs. Le résultat de l'analyse est généralement un ou plusieurs documents, qui contiennent les indications principales sur le fonctionnement désiré du système informatisé. Le document d'analyse contient souvent déjà des prototypes de certains documents importants, que le futur système devra être capable de produire.

Une fois que l'analyse est terminée, il s'agit d'élaborer une série de modèles, basés sur le document d'analyse. Ces modèles nous permettront plus tard d'implémenter une base de données, qui contiendra toutes les informations nécessaires au bon fonctionnement du système informatisé.

**La création de ces modèles se fait selon une certaine méthode**. Nous allons baser notre cours sur la méthode **MERISE** (Méthode d'Etude et de Réalisation Informatique de Systèmes d'Entreprise), qui a été développée pendant les années '70 sous l'impulsion du ministère français de l'industrie. Merise est aujourd'hui largement répandue au Luxembourg, mais également dans beaucoup d'autres pays européens.

**MERISE prévoit une conception par niveaux, et définit pour cela 3 niveaux essentiels:**

1. Le **niveau conceptuel**, qui se base directement sur l'analyse, décrit l'ensemble des données du système d'information, sans tenir compte de l'implémentation informatique de ces données. Ce niveau, qui représente donc la signification des données, se traduit par un formalisme que nous appelons:

**Modèle conceptuel des données (MCD)**

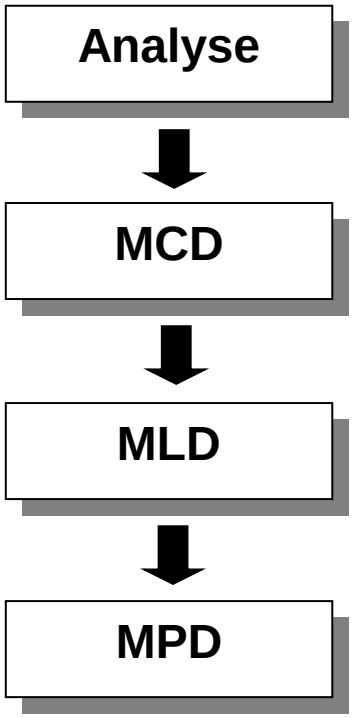
2. Le **niveau logique**, qui se base sur le modèle conceptuel des données, prend en considération l'implémentation du système d'information par un SGBD. Ce niveau introduit la notion des tables logiques, et constitue donc le premier pas vers les tables des SGBD. Ce niveau est représenté par le:

**Modèle logique des données (MLD)**

3. Le **niveau physique**, qui se base sur le modèle logique des données, contient finalement les tables définies à l'aide d'un SGBD spécifique (p.ex. MS Access, dBASE, Oracle …). Ce niveau est représenté par le:

**Modèle physique des données (MPD)**

Voici donc les 4 étapes nécessaires pour traduire un système d'information naturel en une base de données:

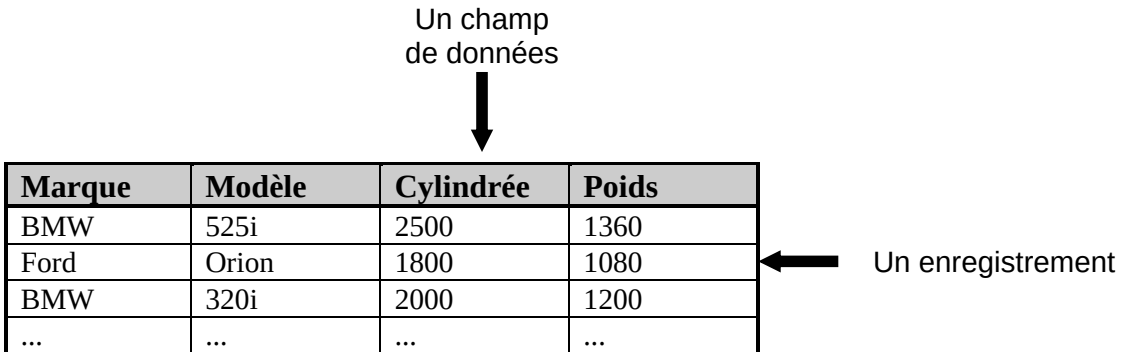


## 3.2 Pourquoi modéliser ?

Nous avons vu qu'une base de données est constituée par un ensemble de tables qui contiennent toutes les données de la base. Une méthode de modélisation nous permet de trouver le bon nombre de tables pour une base de données et de déterminer quelles données sont représentées à l'intérieur de quelle table.

Pour l'instant, il nous suffit de savoir qu'une table est un ensemble d'enregistrements, dont chacun est composé par les mêmes champs de données. On pourrait comparer une table à une liste en MS-Excel[1]. Les tables sont étudiées en détail dans le chapitre 6.

Voici un exemple d'une table :




| Marque | Modèle | Cylindrée | Poids |
|---|---|---|---|
| BMW | 525i | 2500 | 1360 |
| Ford | Orion | 1800 | 1080 |
| BMW | 320i | 2000 | 1200 |
| ... | ... | ... | ... |

A l'aide d'un exemple précis, nous allons voir pourquoi il est important de bien réfléchir sur le nombre de tables d'une base de données et sur la structure de chaque table.

Il s'agit de créer une base de données pour une caisse de maladie. On veut stocker tous les employés-membres de la caisse avec leur société-employeur. Afin de faciliter l'exercice, nous allons uniquement stocker les informations suivantes pour chaque employé:

- le numéro de l'employé
- le nom de l'employé
- le prénom de l'employé
- le numéro de son entreprise
- le nom de son entreprise
- la localité où se trouve l'entreprise

A première vue, la solution suivante s'impose :

| NoEmp | Nom_Emp | Prénom_Emp | NoEntr | Nom_Entr | Localité |
|---|---|---|---|---|---|
| 102 | Boesch | Emil | 1 | Schaffgaer S.à r.l. | Differdange |
| 103 | Midd | Erny | 2 | Gudjär | Colmar Berg |
| 104 | Witz | Evelyne | 1 | Schaffgaer S.à r.l. | Differdange |
| 105 | Kuhl | Menn | 1 | Schaffgaer S.à r.l. | Differdange |
| 106 | Super | Jhemp | 2 | Gudjär | Colmar Berg |
| ... | ... | ... | ... | ... | ... |

[1] voir cours de la classe 12CG

Nous voyons ici uniquement quelques enregistrements. Une caisse de maladie ayant des miliers de membres, et cette table possédant un enregistrement par membre, on peut bien s'imaginer la taille réelle de la table.

Hors cette solution, bien qu'elle soit correcte dans le sens le plus large du terme, nous impose un certain nombre de problèmes .

## Exercice 1

Essayez de trouver en discussion quelques problèmes qui peuvent se manifester lors du travail journalier avec cette table.

## Exercice 2

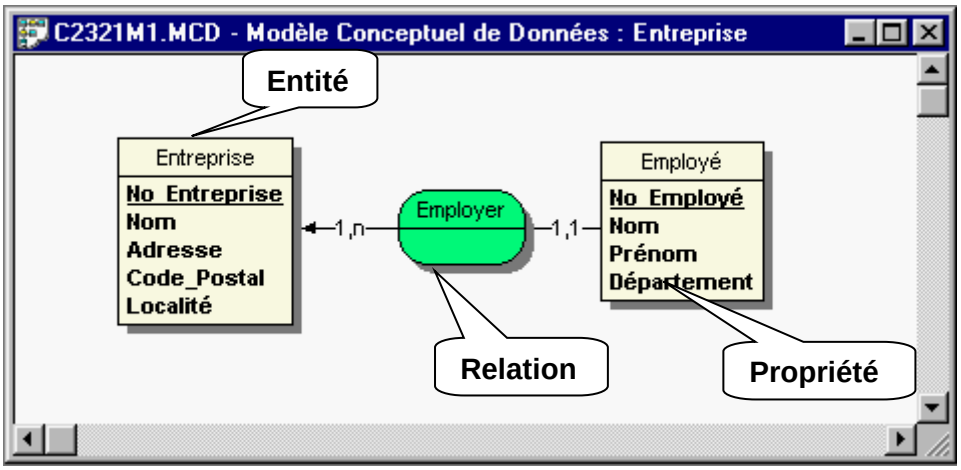
Comment est-ce qu'on pourrait éviter ces problèmes sans toutefois perdre des informations ?

# 3.3 Le modèle conceptuel des données (MCD)

## 3.3.1 Définition

En se basant sur un document d'analyse, le modèle conceptuel des données (MCD) fait référence à tous les objets du système d'information et à des relations entre ces objets. Le formalisme utilisé dans ce modèle est encore connu sous le nom de "Schéma Entité-Relation". Ce formalisme se base autour de 3 concepts principaux, les **entités**, les **relations** et les **propriétés**.

Voici par exemple un MCD qui représente une entreprise avec ses employés.




Nous allons par la suite détailler le rôle de ces 3 concepts de base du MCD.

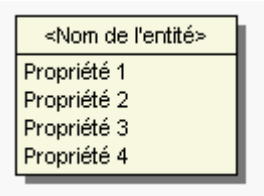
## 3.3.2 La notion d'entité

**Une entité permet de modéliser un ensemble d'objets concrets ou abstraits de même nature.**

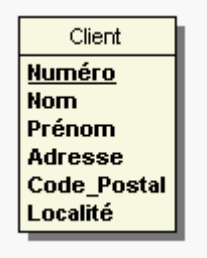
Dans l'exemple du chapitre précédent , l'entité *Entreprise* spécifie donc l'ensemble des entreprises, qui nous intéressent dans le contexte de notre système d'information. De même, l'entité *Employés* représente tous les employés de notre système d'information.

**Une entité est caractérisée par son nom et ses propriétés**.

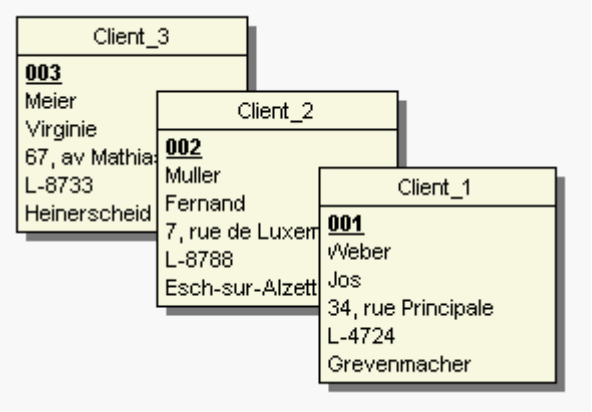
Représentation graphique:




Prenons par exemple une entité *Client*:




Voici quelques exemples de clients:




**Chacun de ces clients représente une occurrence de l'entité *Client*.**

### 3.3.3 La notion de propriété

**Une propriété est une donnée élémentaire d'une entité.**

**Une propriété est unique dans un MCD; et ne peut pas être rattachée à plusieurs entités différentes.**

Représentation graphique d'une propriété:
Le nom de la propriété est indiqué à l'intérieur du rectangle qui représente l'entité correspondante.

Voici quelques exemples de propriétés:

Pour une entité *Client*:
- Nom du client
- No.Tél. du client

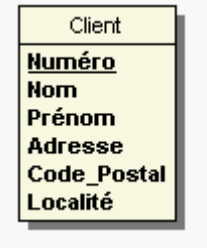
Pour une entité *Salarié*:
- Nom du salarié
- No. Matricule
- Salaire mensuel

Pour une entité *Contrat d'assurance*:
- No Contrat
- Type d'assurance
- Montant assuré

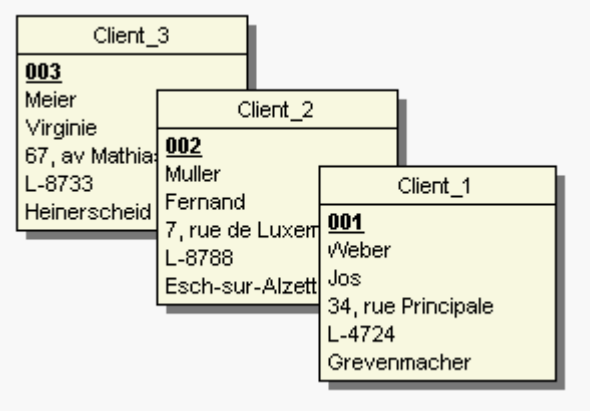
**A l'intérieur des occurrences, les propriétés prennent des valeurs**

Exemple:

L'entité *Client* est définie par les propriétés suivantes:




A l'intérieur de chaque occurrence, chaque propriété prend une valeur, qui est dans la plupart des cas une **valeur numérique**, une valeur sous forme de **texte** ou encore une **date**.




La propriété **Nom** prend p.ex. les valeurs "Meier", "Muller" et "Weber" dans les 3 occurrences.

**A l'intérieur de chaque occurrence, chaque propriété ne prend qu'une seule valeur au maximum.**

Le client 002 par exemple ne peut pas avoir 2 adresses.

### 3.3.4 La notion d'identifiant

**Afin de pouvoir distinguer les différentes occurrences d'une même entité, l'entité doit être dotée d'un identifiant. L'identifiant est composé d'une ou de plusieurs propriétés de l'entité. Chaque occurrence d'une entité doit avoir une valeur différente pour l'identifiant** Le choix d'un identifiant correcte est très important pour la modélisation:

Comme choix pour l'identifiant d'une entité nous distinguons généralement 3 possibilités:

1. Une propriété **naturelle**
   Exemple: Le nom d'un pays pour une entité *Pays*

2. Une propriété **artificielle** qui est inventée par le créateur du MCD
   Exemple: Le numéro d'un client pour une entité *Client*

3. Une propriété **composée** d'autres propriétés naturelles
   Exemple: Le nom et la localité pour une entité *Entreprise*

**Représentation graphique de l'identifiant d'une entité:**
La ou les propriétés qui constituent l'identifiant d'une entité sont soulignées.

**Exercice**

Indiquez graphiquement les entités qui représentent :

1. les passagers d'un vol d'une société aérienne. Nous supposons que la société garde ces informations après le vol ;

2. les résultats sportifs de l'entrainement d'un coureur ;

3. les médicaments d'une pharmacie.

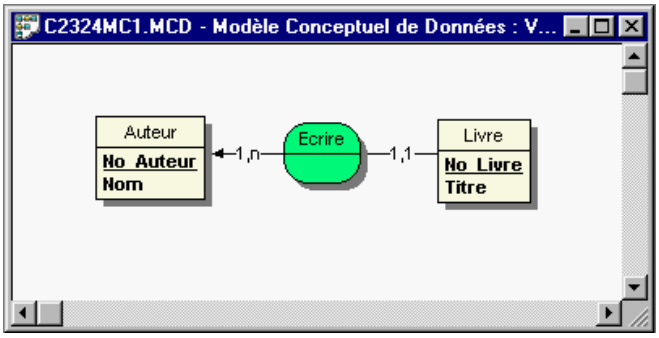
## 3.3.5 La notion de relation

### *3.3.5.1 Définition*

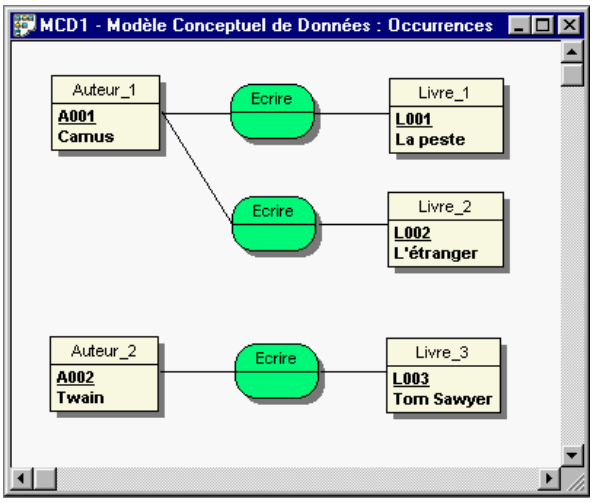
**Une relation[1] décrit un lien entre deux ou plusieurs entités.**
**Chaque relation possède un nom, généralement un verbe à l'infinitif.**
**Bien qu'une relation n'ait pas d'identifiant propre, elle est implicitement identifiée par les identifiants des entités auxquelles elle est liée.**

**Nous distinguons deux types de relations:**
- **les relations binaires, qui sont liées à 2 entités;**
- **les relations ternaires, qui sont liées à 3 entités.**

Exemple d'une relation binaire:




L'occurrence d'une relation est représentée par les occurrences des entités liées à la relation. Voici quelques occurrences de la relation *Ecrire*.



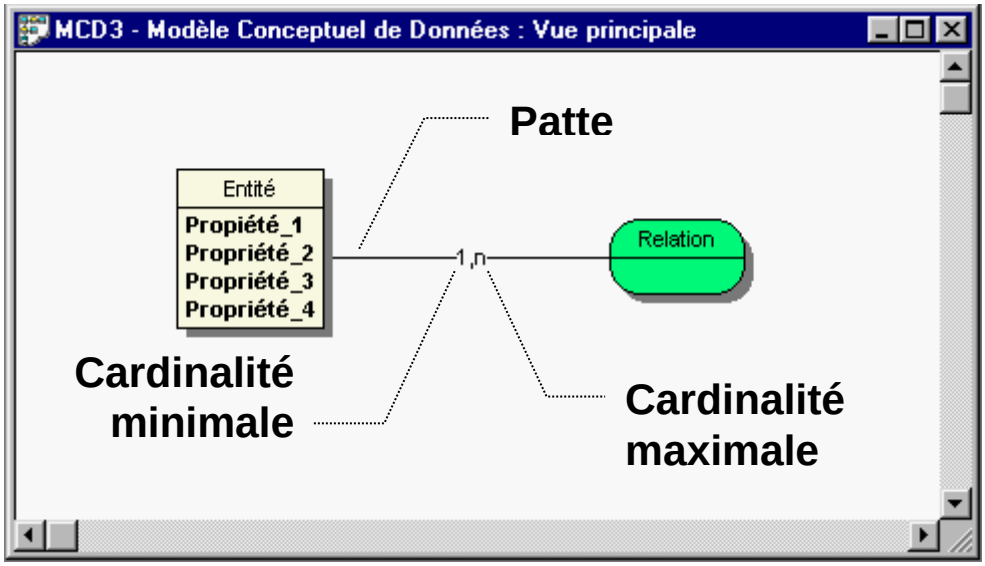

Une occurrence d'une relation est uniquement déterminée par les occurrences des entités liées à la relation.

[1] dans certains livres on parle encore d'une **association** pour désigner une relation

**Pour chaque occurrence d'une relation, l'identifiant composé des identifiants des entités liées à la relation doit être unique.**

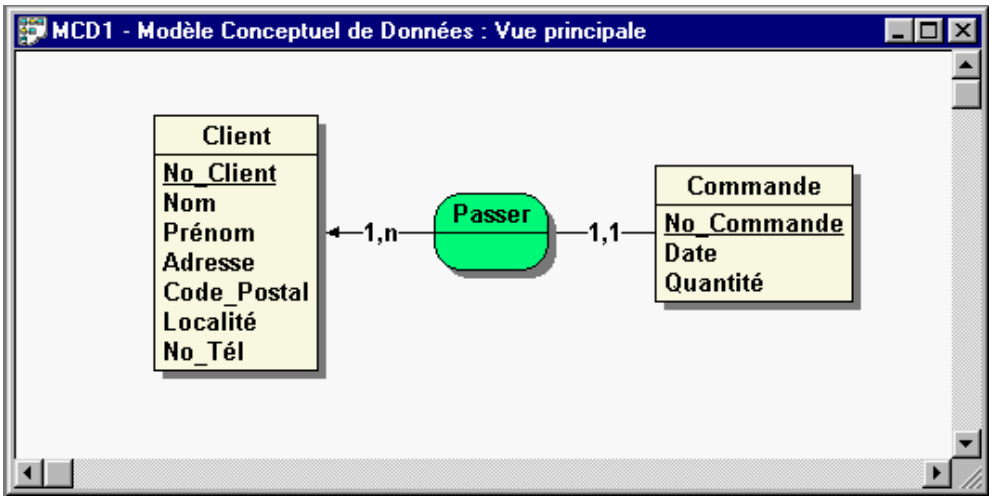
### *3.3.5.2 Les cardinalités d'une relation*

**Une relation est liée à chacune de ses entités par une patte. Sur la patte, on indique les cardinalités.**
**Les cardinalités précisent la participation de l'entité concernée à la relation. Le premier nombre indique la cardinalité minimale, le deuxième la cardinalité maximale.**




Quelle est la signification des cardinalités ?

Exemple 1:



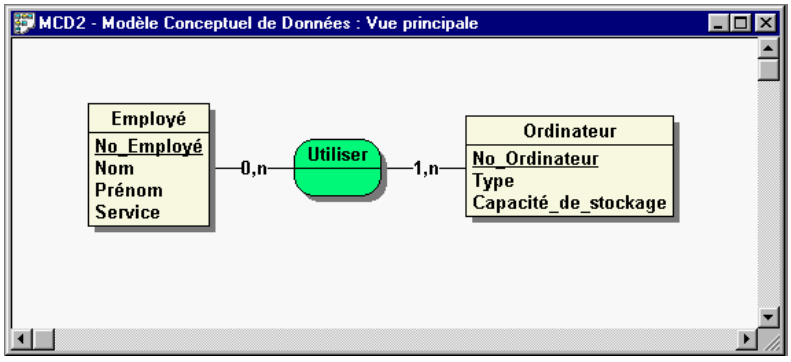
Dans le MCD précédent, entre l'entité *Client* et la relation *Passer*, nous avons les cardinalités suivantes:

- Cardinalité minimale = 1 , *ce qui veut dire que chaque client passe au moins une commande.*
- Cardinalité maximale = n , *ce qui veut dire que chaque client peut passer plusieurs (n) commandes.*

Entre l'entité *Commande* et la relation *Passer*, nous retrouvons les cardinalités suivantes:

- Cardinalité minimale = 1 , *donc chaque commande est passée par au moins un client.*
- Cardinalité maximale =1 , *chaque commande est passée au maximum par un seul client*.

Exemple 2:




Entre l'entité *Employé* et la relation *Utiliser*, nous avons:

- Cardinalité minimale = 0 :
  *Certains employés n'utilisent pas d'ordinateur*

- Cardinalité maximale = n:

---

Entre l'entité Ordinateur et la relation Utiliser, nous avons:

- Cardinalité minimale = 1 :

---

- Cardinalité maximale = n :

---

**De façon générale, on peut dire:**

**La cardinalité minimale exprime le nombre minimum de fois q'une occurrence d'une entité participe à une relation. Cette cardinalité est généralement 0 ou 1.**

- Cardinalité minimale = 0 : Certaines occurrences de l'entité ne participent pas à la relation
- Cardinalité minimale = 1 : Chaque occurrence de l'entité participe au moins une fois à la relation

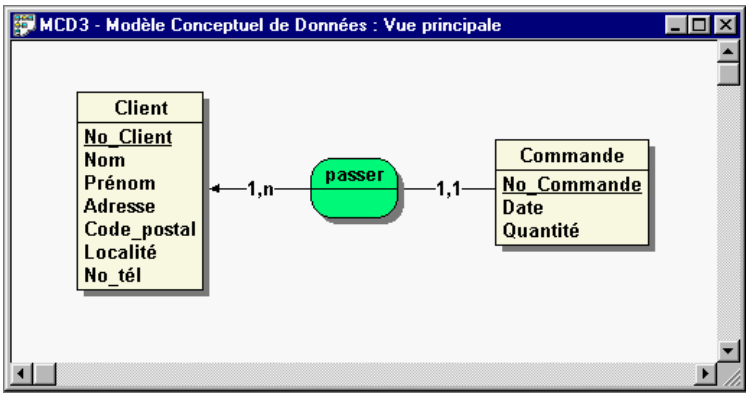
**La cardinalité maximale exprime le nombre maximum de fois q'une occurrence d'une entité participe à une relation. Cette cardinalité vaut souvent 1 ou n, avec n indiquant une valeur >1 mais pas connue à priori.**

- Cardinalité maximale = 1 : Chaque occurrence de l'entité participe au maximum une seule fois à la relation.
- Cardinalité maximale = n : Chaque occurrence de l'entité peut participer plusieurs fois à la relation.

En pratique, afin de déterminer les bonnes cardinalités, le concepteur doit se référer aux résultats de l'analyse.

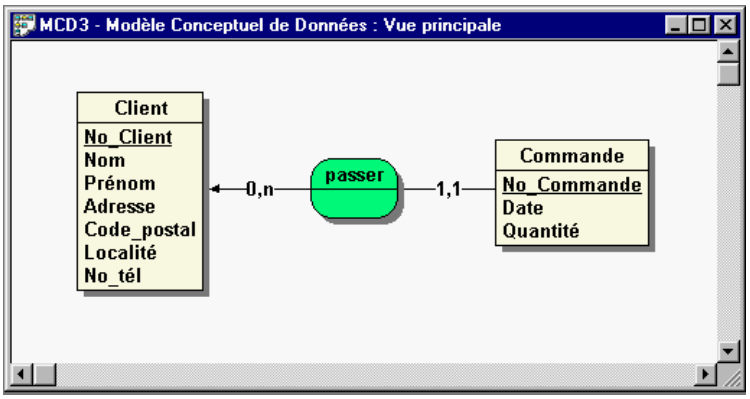
Exemple 3:

Pour les deux cas suivants, on peut affirmer qu'une commande est toujours passée par au moins un client. Une commande est également passée au maximum par un client. Une commande est donc toujours passée par un et un seul client.




Un client passe **au moins une** commande et **au maximum plusieurs (n)** commandes.

Cette modélisation ne tient pas compte des clients qui ne passent aucune commande. Un client est uniquement considéré comme tel s'il passe au moins une commande.



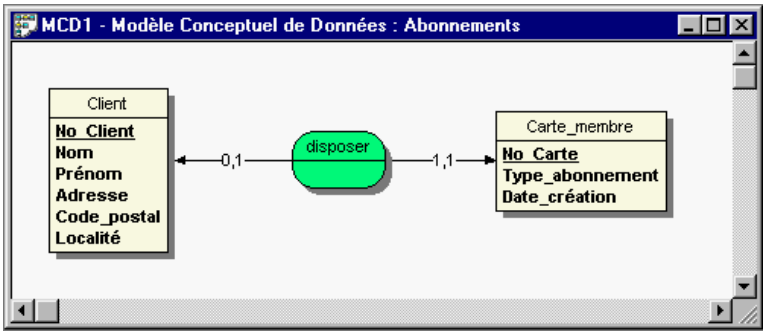

Un client **peut passer aucune** commande et **au maximum plusieurs (n)** commandes.

Cette modélisation tient compte des clients qui ne passent aucune commande.

Laquelle des deux modélisations est correcte ?

Les deux modélisations sont bien sûr correctes. Le choix dépend uniquement du résultat de l'analyse. Imaginez que vous avez fait pendant l'analyse une interview avec un futur utilisateur du système. Il vous a dit qu'il veut enregistrer dans son système des clients potentiels, c.à.d. des personnes dont la compagnie dispose des données personnelles, mais qui n'ont encore jamais passé de commande auprès de la compagnie. La compagnie désire enregistrer les données de ces personnes afin de leur envoyer de temps en temps des publicités. Dans ce cas vous devez opter pour la deuxième modélisation.

Exemple 4:



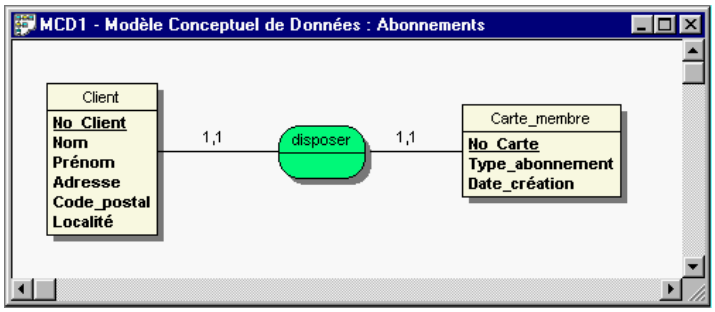

Interprétez cette modélisation :

On dit que *Client* est **l'entité indépendante** par rapport à la relation *disposer* (cardinalité minimale = 0) , tandis que *Carte_membre* est **l'entité dépendante** par rapport à la relation *disposer* (cardinalité minimale = 1).
Une occurrence d'un client peut donc très bien exister sans carte de membre, mais une carte de membre ne peut jamais exister sans client. **La cardinalité minimale nous indique donc si une entité est indépendante ou dépendante.**

**On dit qu'une entité est indépendante par rapport à une relation lorsque sa cardinalité minimale vaut 0, et dépendante par rapport à une relation lorsque sa cardinalité minimale vaut 1.**

**Une relation ne peut pas être liée uniquement à des entités dépendantes ayant en plus une cardinalité maximale de 1 ! ! !**

La modélisation suivante par exemple n'est pas correcte:

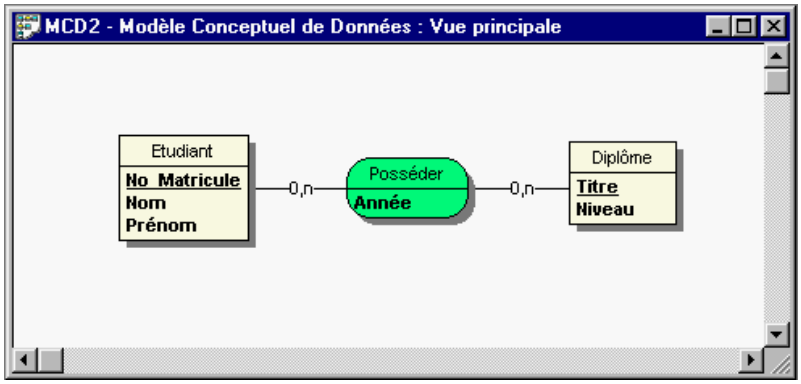



Dans ce cas, il faut réunir les propriétés des deux entités dans une seule.

### 3.3.5.3 Propriétés d'une relation

Une relation peut généralement être dotée de propriétés.

Exemple:

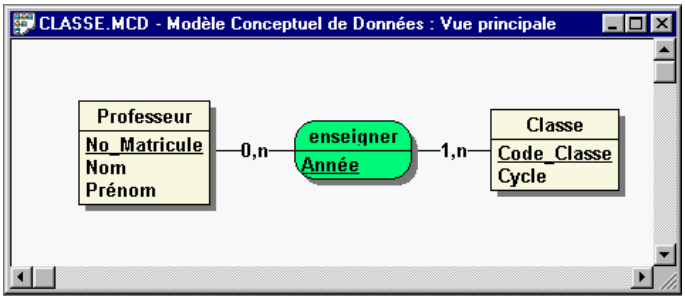



**Exercice**

Pourquoi est-ce qu'on ne peut pas associer la propriété *Année* à une des entités ?

**Attention: Cette propriété peut même devenir une partie de l'identifiant. Dans ce cas, elle doit être soulignée.**

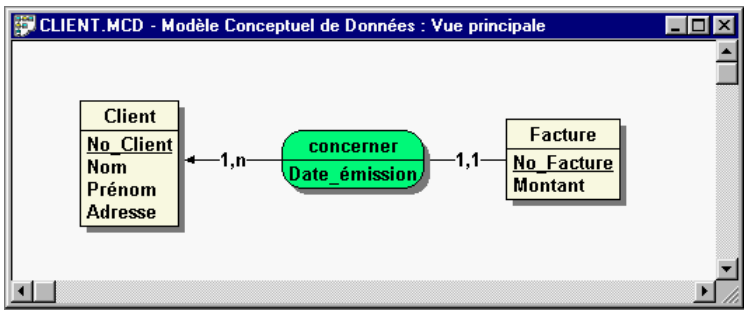
Exemple:




Comme un professeur peut avoir la même classe pendant plusieurs années (p.ex. Jos Weber → 12CG2), un identifiant composé de *No_Matricule* et *Code_Classe* n'est pas suffisant, puisqu'il ne garantit pas l'unicité. On y ajoute l'*Année*.

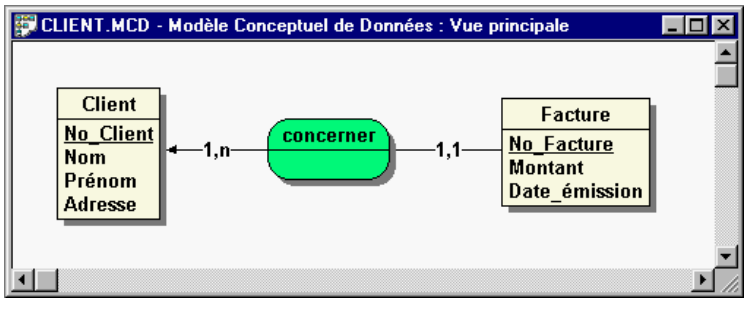
**Attention: Une relation à cardinalité (1,1) n'est jamais porteuse de propriétés. Dans ce cas, les propriétés migrent dans l'entité portant cette cardinalité (1,1).**

Exemple:




Cette modélisation n'est **pas correcte** ! Chaque facture ne possède qu'une et une seule date d'émission, ce qui fait que la propriété *Date_émission* doit migrer dans l'entité *Facture*.

Voici la modélisation correcte:



## 3.3.6 Exemple "KaafKaaf"

PARTIE 1

La société "KaafKaaf" désire informatiser son système de facturation. Les factures devraient se présenter de la façon suivante:

**"KaafKaaf" S.à r.l.**
**12, avenue Goss**
**L-1234 Luxusbuerg**

**FACTURE No. 12345**

Luxusbuerg, le 31.08.1997

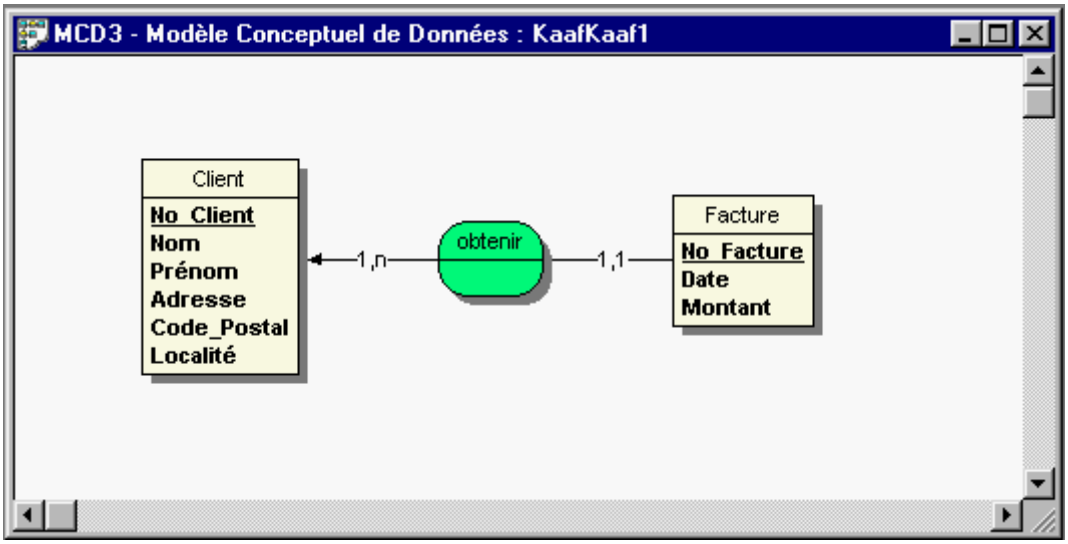
**Client**

| | |
|---|---|
| Nom: | **Weber** |
| Prénom: | **Jos** |
| Adresse: | **23, rue Principale** |
| Code_postal: | **L-3456** |
| Localité: | **Grevenmacher** |

**Montant de la facture:** **3400** Luf.

Créez un MCD, qui permet de modéliser correctement le système d'information nécessaire, sachant que:

- Un client peut bien sûr recevoir plusieurs factures, mais il est uniquement considéré comme tel à partir du moment où il reçoit sa première facture.
- Une facture concerne un et un seul client.



Remarque:

Bien que le numéro du client n'apparaisse pas en tant que tel sur la facture, il est préférable d'ajouter cette propriété artificielle à l'entité *Client*, et de la définir comme identifiant de cette entité. Cela nous empêche de devoir définir un identifiant composé de trop de propriétés.

## PARTIE 2

Il s'agit d'étendre le MCD de la partie 1.

Le responsable de la facturation de la société désire rendre les factures plus informatives. Comme un client peut acheter plusieurs articles différents en même temps, la facture devrait indiquer pour chaque article le numéro , un libellé, le prix unitaire, la quantité vendue et le prix total pour ce type d'article.

Voici l'aspect que la facture devrait avoir:

**"KaafKaaf" S.à.r.l.**
**12, avenue Goss**
**L-1234 Luxusbuerg**

**FACTURE No. 12345**

Luxusbuerg, le 31.08.1997

**Client**

| | |
|---|---|
| Nom: | **Weber** |
| Prénom: | **Jos** |
| Adresse: | **23, rue Principale** |
| Code_postal: | **L-3456** |
| Localité: | **Grevenmacher** |

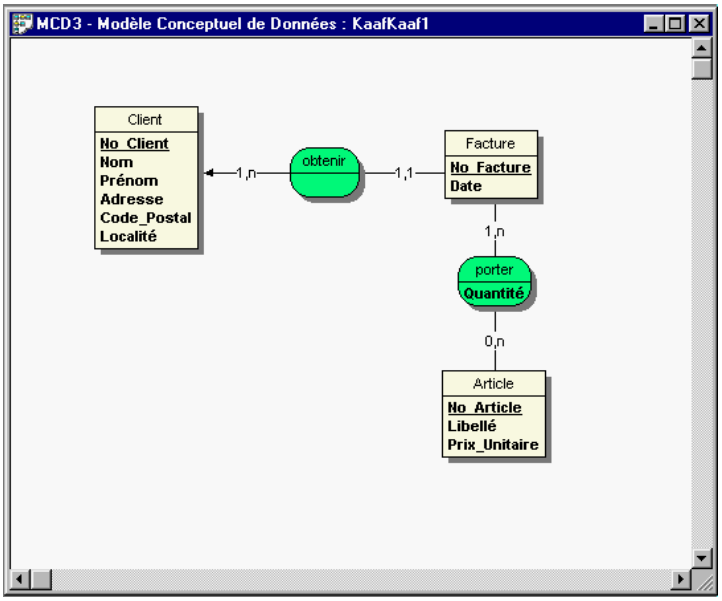
**Articles:**

| No. Article | Libellé | Prix unitaire | Quantité | Prix |
|---|---|---|---|---|
| 234 | Marteau | 470 Luf. | 1 | 470 Luf. |
| 566 | Tournevis | 220 Luf. | 3 | 660 Luf. |
| 023 | Pince à tuyaux | 800 Luf. | 1 | 800 Luf. |

**Montant total de la facture:** **1930 Luf.**

Proposez un nouveau MCD qui reflète ces modifications, en respectant que:

- Tous les articles disponibles sont stockés (p.ex. No=234 Libellé="Marteau" PU=470 Luf.). Même si un article n'est pas encore considéré par une facture, il existe dans le système d'information.



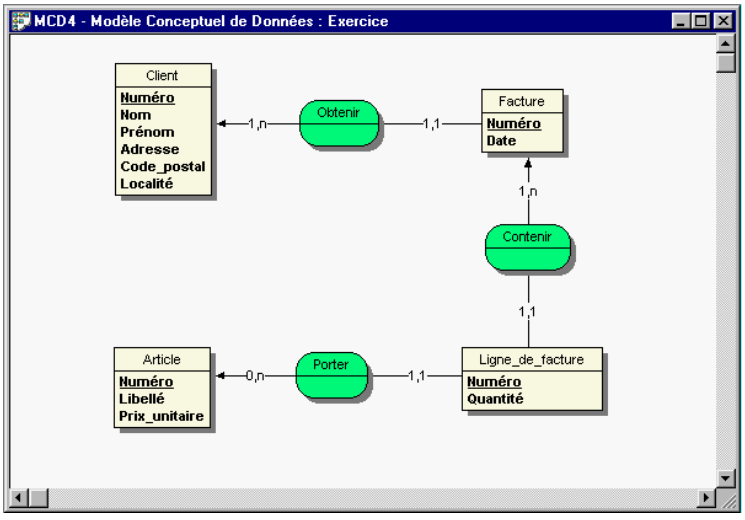

Remarques:

- L'entité *Facture* ne contient plus la propriété *Montant*. Il existe une règle générale de conception qui dit:

  **Aucune propriété qui peut être calculée à partir d'autres propriétés existantes, ne devra être stockée dans le MCD.**

  Pour la même raison, on n'a pas besoin de modéliser explicitement le prix à payer pour l'achat d'une quantité d'articles donnés. Le prix pour chaque article figurant sur la facture peut être calculé à partir du prix unitaire et de la quantité

- Nous retrouvons ici le cas d'une relation qui a une propriété. En fait, la propriété *Quantité* n'est pas spécifique à un article, mais à l'achat de cet article à l'aide d'une facture. Cette façon de modéliser la situation est la plus facile, mais il existe une alternative. On peut introduire l'entité abstraite *Ligne_de_facture*, qui représente une ligne de détail d'une facture, p.ex celle pour le marteau.



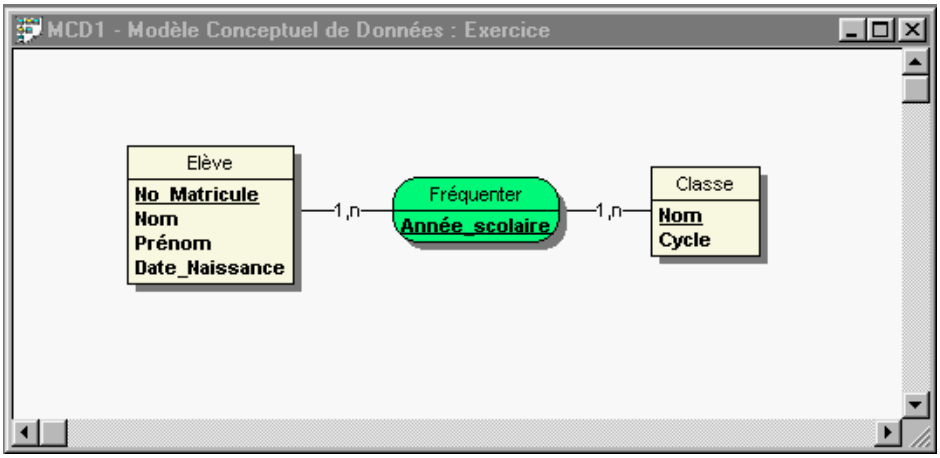
## 3.3.7 Exemple "Gestion d'école"

PARTIE 1

Dans une école, on veut informatiser le système d'information qui gère les classes.

Elaborez un MCD sachant que:

- Un élève est caractérisé par son no. matricule, son nom et prénom, ainsi que sa date de naissance.
- Une classe est caractérisée par le nom de la classe (p.ex 13CG2) et par une indication du cycle (valeurs possibles: "inférieur", "moyen", "supérieur").
- Il faudra prévoir de connaître la fréquentation des classes des élèves sur plusieurs années consécutives.
- Un élève enregistré dans le système fréquente au moins une classe au cours des années.

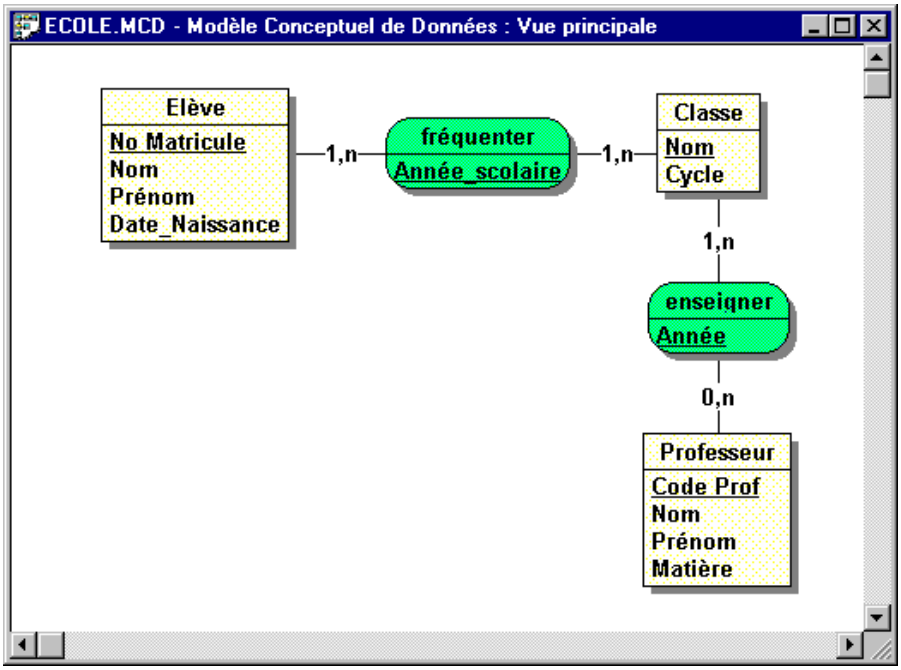



PARTIE 2

Il s'agit maintenant de concevoir une extension au MCD précédent qui permet de représenter la situation suivante:

- La direction de l'école désire également saisir tous les professeurs dans le système d'information. Un professeur est caractérisé par un code interne unique (p.ex. Jemp Muller aura le code JEMU), son nom et prénom et la matière qu'il enseigne**. Nous supposons que chaque professeur enseigne une seule matière**.

- Modélisez le fait que chaque classe est enseignée chaque année par un ou plusieurs enseignants. Un enseignant peut bien sûr donner des cours dans plusieurs classes, mais peut également ne pas donner des cours pendant une ou plusieurs années.


ECOLE.MCD - Modèle Conceptuel de Données : Vue principale
Elève
No Matricule
Nom
Prénom
Date_Naissance
1,n
fréquenter
Année_scolaire
1,n
Classe
Nom
Cycle
1,n
enseigner
Année
0,n
Professeur
Code Prof
Nom
Prénom
Matière

## 3.3.8 L'utilisation d'une relation ternaire

Lors de l'introduction des relations nous avons déjà mentionné la notion de **relation ternaire**. Une relation ternaire est une relation à laquelle sont liée 3 entités.

Bien que dans la pratique la plupart des relations soient binaires (2 entités) il existe cependant des situations où l'utilisation d'une relation ternaire s'impose.

Exemple :

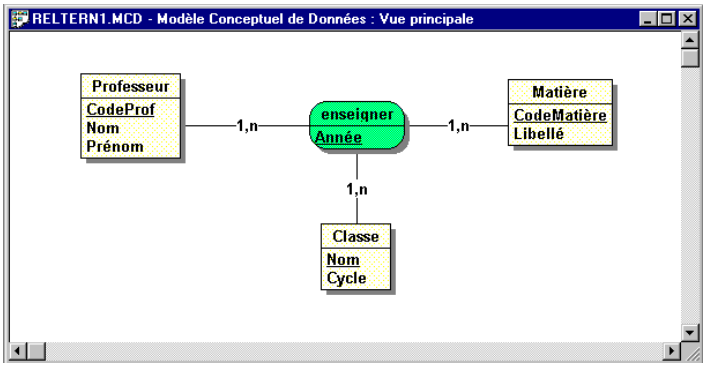
A partir des 3 entités ***Professeur*** *(CodeProf, Nom, Prénom);* ***Matière****(CodeMatière, Libellé)* et ***Classe****(Nom,Cycle)* il s'agit de créer un MCD qui renseigne sur le fait quelle matière est enseignée dans quelle classe par quel professeur pour une année scolaire donnée.

**Exercice**

Essayez de montrer les limites/défauts d'un MCD qui représente l'énoncé de l'exemple précédent en utilisant uniquement des relations binaires.

Solution de l'exemple précédent :

Voici une solution qui utilise une relation ternaire




Il existe 3 façons pour lire/interpréter ce modèle:

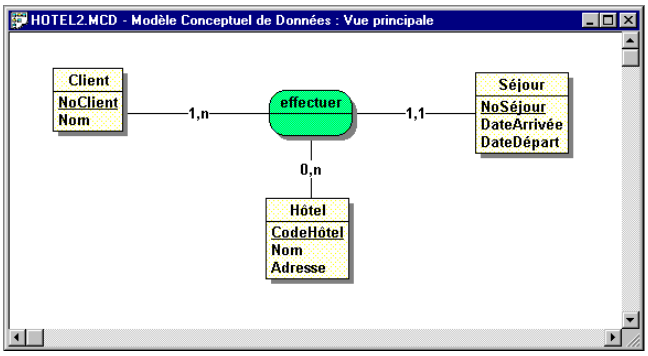
- Un professeur peut enseigner 1 à n fois **une matière dans une classe.**
- Une matière peut être enseignée 1 à n fois **par un professeur dans une classe.**
- Une classe peut être enseignée 1 à n fois **dans une matière par un professeur.**

On peut dire que chaque occurrence de la relation *enseigner* associe un professeur à une matière et une classe pour une année donnée. Ou encore, ce modèle nous permet de montrer pour chaque année scolaire quelle matière est enseignée dans quelle classe par quel professseur.

Il n'est pas toujours facile de déterminer quand il faut utiliser une relation ternaire. Généralement, on peut déjà affirmer que si une ou plusieurs des entités liées à une relation ternaire possèdent une cardinalité maximale de 1, la modélisation n'est pas optimisée dans le sens qu'il faudrait mieux décomposer la relation ternaire, c.à.d. la représenter par 2 relations binaires.

Exemple :

La direction d'une chaîne d'hôtels désire gérer les séjours des clients dans les différents hôtels. Comme on peut effectivement dire "*Un client effectue un séjour dans un hôtel*" on est ammené à proposer la modélisation suivante.

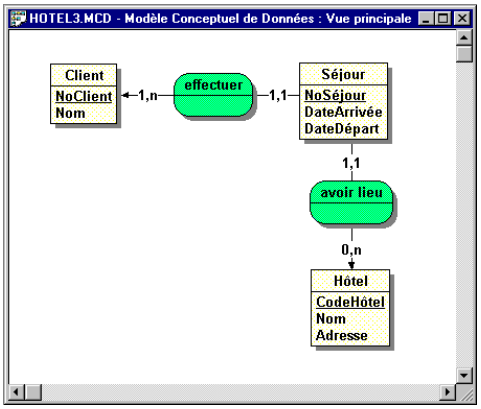



Il existe 3 façons pour lire/interpréter ce modèle:

- Un client peut effectuer 1 à n fois **un séjour dans un hôtel**.
- Dans un hôtel peut être effectué 0 à n fois **un séjour par un client.**
- Un séjour peut être effectué une et une seule fois **par un client dans un hôtel**.

Chaque occurrence de la relation *effectuer* associe donc un séjour à un client et à un hôtel.

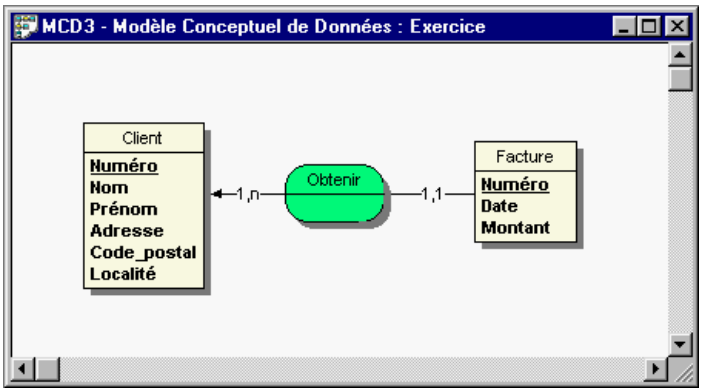
Hors, cette modélisation porte une contrainte supplémentaire, puisque la cardinalité 1,1 entre l'entité *Séjour* et la relation nous indique que pour chaque occurrence de *Séjour* il ne peut exister qu'une et une seule occurrence de la relation. Donc chaque séjour est associé une et une seule fois à une combinaison client/hôtel. Dans ce cas il vaut mieux décomposer la relation ternaire de la façon suivante:



### 3.3.9 Les contraintes d'intégrité fonctionnelle (CIF)

**Quand on détermine entre une relation et une entité une cardinalité qui présente les valeurs 0,1 ou 1,1, alors cette relation est particulière et on dit qu'elle représente une Contrainte d'Intégrité Fonctionnelle (CIF).**

Exemple:



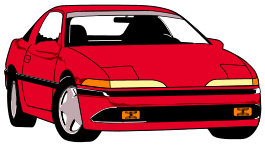

La relation *Obtenir* représente une CIF.

Une CIF indique que l'une des entités est totalement déterminée par la connaissance de l'autre. Dans notre exemple on peut dire que connaissant une facture bien précise, on connaît avec certitude le client correspondant.

Comment est-ce qu'on représente une CIF dans un MCD ?

Une CIF est représentée par une **flèche** sur la patte opposée à celle ayant une cardinalité 0,1 ou 1,1. L'entité qui est attachée à cette patte est appelée entité **cible** de la CIF, tandis que l'autre entité constitue l'entité **émettrice** de la CIF.

## 3.3.10 Exercices



### Exercice 1

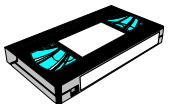
Voici le résultat simplifié d'une analyse faite auprès d'une compagnie d'assurance qui désire informatiser la gestion des contrats auto.

- Un client peut assurer plusieurs voitures auprès de la compagnie. Chaque voiture est assurée par un seul contrat. Un contrat assure une seule voiture.
- En ce qui concerne un client, la compagnie désire connaître son nom, prénom, adresse complète, numéro de téléphone ainsi qu'un numéro de compte bancaire avec indication de la banque.
- Chaque contrat contient un numéro de contrat unique, la prime annuelle à payer, la date de paiement annuel, la marque de la voiture, le modèle de la voiture, le numéro d'immatriculation de la voiture, la valeur de la voiture et la date d'acquisition de la voiture.

En ignorant la méthode de modélisation, on pourrait créer une BD avec une seule table ayant un champ pour chaque donnée indiquée dans l'analyse. On aurait donc les données des clients et des contrats dans une seule table. Quelles en seraient les inconvénients ?

Créez le modèle conceptuel des données correspondant à cette situation

## Exercice 2



Le responsable d'un magasin de location de cassettes vidéo désire informatiser le système de gestion des locations. Voici les informations recueillies pendant l'analyse:

- Un membre est caractérisé par son numéro membre, son nom, son prénom, son adresse ainsi que sa date de naissance. Dès que la carte de membre est payée (paiement unique), le membre est enregistré dans le système et il peut désormais louer des cassettes vidéo.
- Un film est caractérisé par un numéro film, un titre, un code langue, un code genre et une durée. Le même film peut être disponible en plusieurs exemplaires. Un exemplaire est déterminé par un numéro exemplaire ainsi que la date d'achat de l'exemplaire.
- Lors d'une location, un membre peut louer plusieurs films en même temps. En principe, une location a une durée d'un jour, mais cette durée peut être prolongée. Nous supposons qu'un membre rend tous les exemplaires loués en une seule fois.

Créez le modèle conceptuel des données

## Exercice 3

Afin d'informatiser la gestion des séances du cinéma **Limelight**, vous disposez des informations suivantes.

- Un film est enregistré dans le système d'information dès que la (les) copie(s) sont arrivées au cinéma. A partir de ce moment, on commence à programmer des séances pour le film en question. Comme le même film n'est jamais joué dans deux séances parallèles, on peut ignorer la gestion des copies.
- Un film est représenté par un numéro courant interne, qui lui est affecté par le gestionnaire des séances. En plus, on s'intéresse pour le titre, la langue et la durée du film. Lorsqu'un film apparaît en plusieurs langues différentes, on crée dans le système d'information simplement un enregistrement par langue.
- Chaque film est accompagné en général d'une fiche technique, qui renseigne en outre sur le système son du film (p.ex. DOLBY, THX etc.). Cette information est importante, puisque les capacités en ce qui concerne la reproduction du son varient d'une salle dans une autre. Une salle peut supporter plusieurs systèmes différents, tandis qu'un film est tourné en utilisant un seul système son. Un système son est caractérisé par un code identificateur ainsi qu'un libellé.
- Le cinéma dispose actuellement de 12 salles, avec 3 nouvelles salles en construction. Une salle est prise en compte dans le système d'information, dès qu'elle est prête pour accueillir des séances. Une salle est caractérisée par son numéro, sa capacité ainsi que des informations concernant le support des différents systèmes son.
- Le système d'information doit permettre de vendre des tickets pour une séance donnée, même plusieurs jours en avance. La réservation des sièges n'étant pas demandée, il est toutefois nécessaire que le système soit capable de prévenir un excès de la capacité d'une salle en ce qui concerne le nombre de tickets vendus.
- La gestion des prix pour les tickets se fait au niveau des séances, puisque le prix pour voir un même film peut varier d'une séance à une autre (p.ex. Tarif réduit les lundis à 16h00).
- Une séance, qui se déroule évidemment sans une seule salle, est identifiée par un numéro courant.

Créez le modèle conceptuel des données

## Exercice 4

Un club de vente de livres par correspondance propose à ses membres l'achat d'un ou de plusieurs livres via des bons de commandes. Pour cela, des bons de commandes ainsi qu'un catalogue sont envoyés à tous les membres deux fois par an.
Le responsable du club désire informatiser la gestion des commandes de livres. Voici à titre d'exemple un bon de commande:

# Bicherwuerm S.à r.l.

## Commande de livres

Votre numéro membre : **123578**

∗ Veuillez nous indiquer des changements éventuels de vos coordonnées personnelles.

Nom: _____ Prénom: _____
Adresse: _____ CP: _____
Localité: _____ No. Téléphone: _____

Votre commande :

∗ Indiquez s.v.p. pour chaque livre le numéro ISBN et le titre (voir catalogue).

| | Numéro ISBN | Titre |
|---|---|---|
| **1** | | |
| **2** | | |
| **3** | | |
| **4** | | |
| **5** | | |

Cher membre

Les livres commandés vous seront envoyés le plus vite possible. Une facture vous parviendra après livraison complète.

- Au moment où un membre retourne un bon de commande, une nouvelle commande est créée dans le système. Une commande est identifiée par un numéro de commande et caractérisée en plus par une date de commande. Les livres disponibles de la commande sont envoyés tout de suite au membre.
- Pour devenir membre du club, il suffit d'effectuer une fois une commande via des bons de commandes légèrement modifiés (p.ex. pas de numéro de membre), et d'indiquer les coordonnées personnelles.
- Tous les livres présents dans le catalogue sont également stockés dans le système. Un livre est identifié par son numéro ISBN et caractérisé en plus par un titre, un genre (p.ex. ROMAN, HISTOIRE, BIOGRAPHIE, TECHNIQUE ...), une langue (p.ex. ALL,

FRA ...), le nom de l'auteur, le prix de vente et le nombre d'exemplaires disponibles en stock.

- Une facture est identifiée par un numéro de facture et caractérisée en plus par une date de facture.
- Afin de garantir qu'une facture n'est créée et envoyée au membre qui a effectué la commande correspondante qu'au moment où tous les livres de la commande ont été livrés, le système doit être capable de vous informer quels livres de la commande ont déjà été envoyés au membre et quels livres n'ont pas encore été envoyés au membre (p.ex. livre non disponible en stock au moment de la réception de la commande).
- A chaque commande est affectée un seul employé. Un employé est identifié par un code employé (p.ex. WEBJO = Weber Jos). Le responsable du club de vente veut également savoir quel employé (Nom & Prénom) traite quelle commande. Nous supposons que chaque employé a déjà traité des commandes.
- De temps en temps, le responsable de la facturation désire avoir un relevé des factures non encore payées.

Créez le modèle conceptuel des données

## Exercice 5

Le commandant de la brigade municipale des Sapeurs-Pompiers d'Esch-sur-Alzette se propose d'informatiser la gestion des différentes interventions. Etant responsable de l'administration de la brigade, vous êtes en charge de créer une base de données pour stocker les informations relatives aux interventions.

### PARTIE 1

Voici le résultat de l'analyse préliminaire menée auprès des responsables de la brigade (p.ex. le commandant, le sous-commandant …)

- Chaque intervention est identifiée par un numéro unique
- Une intervention est en plus caractérisée par une date, une adresse et un type d'intervention (p.ex. Incendie, Accident, Inondation)
- Pour chaque intervention, on veut savoir quels sapeurs-pompiers y ont participé
- Un sapeur-pompier est caractérisé par un numéro d'identification, son nom, son prénom, son âge ainsi qu'un grade (p.ex. sapeur, chef de section, sous-commandant, commandant)

Travail à réaliser:

Créez le modèle conceptuel des données


### PARTIE 2

Le commandant vous demande de représenter également l'utilisation des véhicules de la brigade. Lors d'une intervention, les sapeurs-pompiers sont affectés aux différents véhicules selon les circonstances. Jusqu'à présent, le commandant a rempli une fiche pour chaque intervention (Exemple: voir page suivante)

- Un véhicule est identifié par son numéro d'immatriculation
- Un véhicule est en plus caractérisé par un type de véhicule (p.ex. Echelle, Transport ...), une marque, et le nombre de places disponibles

Travail à réaliser:

Créez le modèle conceptuel des données

# Service Incendie – Esch-sur-Alzette

## Fiche d'intervention

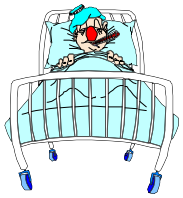
No-Intervention: **00235** Date: **11/11/1997** Type: **Incendie**

Adresse: **12, bvd. Hubert Clement L-4076 Esch-Sur-Alzette**

| Véhicule | Sapeur |
|---|---|
| **Echelle Magirus-Deutz** | **Emilio Pegaso** |
| | **Jang van der Heck** |
| **Camion à double pompe** | **Toto Alnasso** |
| | **Jemp Grisu** |
| **Transport Ford Transit** | **Emil Zweemil** |
| | **Kathrin Allburn** |
| | **Metti Paletti** |
| | **Jacques Guddebuer** |
| | **Hary Beau** |
| | |
| | |
| | |

Signature responsable: _____

## Exercice 6



Il s'agit d'informatiser la gestion des séjours des patients d'un hôpital, ainsi que la gestion des interventions effectuées par les médecins. Jusqu'à présent, cette gestion s'est effectuée à l'aide des fiches suivantes.

# Hôpital Municipal de Heinerscheid

### Gestion des séjours et interventions

| PATIENT |
|---|
| No Matricule: |
| Nom: |
| Prénom: |
| Adresse: |
| Code Postal: |
| Localité: |
| Caisse de maladie: |

| SEJOUR |
|---|
| No Séjour: |
| Date Arrivée: |
| Date Départ: |
| Frais à charge du patient: |
| No Chambre: |
| Etage: |
| Classe: |

**INTERVENTION(S):**

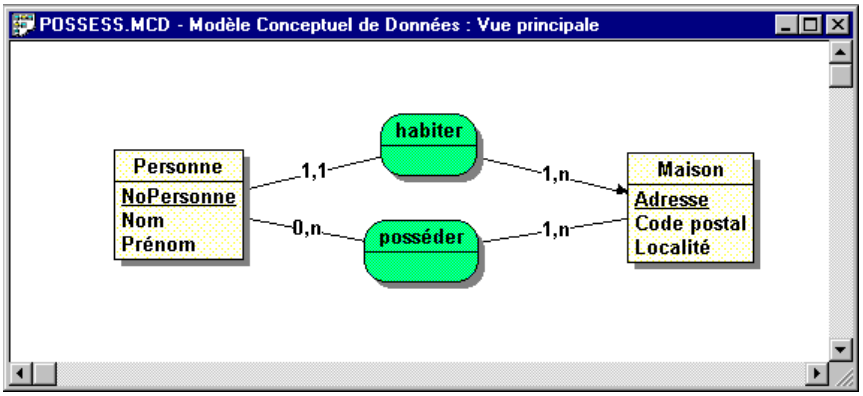
| Code intervention: | Description: | Date: | Code Médecin: | Nom: | Prénom: |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

- Nous supposons qu'un patient occupe la même chambre pendant toute la durée de son séjour.
- A part des informations concernant les médecins, qui se trouvent déjà sur les fiches, on désire stocker dans le système d'information le numéro de téléphone et la spécialité de chaque médecin.
- Les interventions sont identifiées par un code et une description. L'hôpital dispose d'une liste d'interventions prédéfinies. (p.ex. 0236 Tomographie du crâne)
- Les données actuelles sont migrées dans la nouvelle application informatique.

## 3.3.11 Cas particuliers du MCD

### *3.3.11.1 Plusieurs relations différentes entre deux entités*

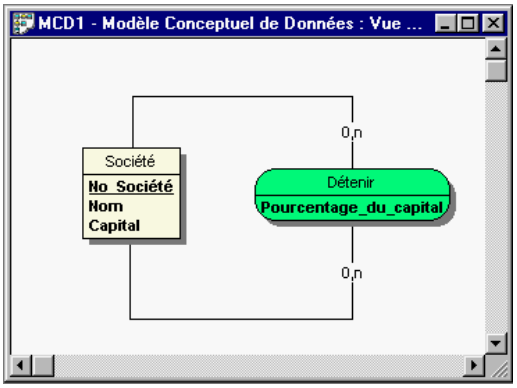
Exemple:




Une personne, qui habite dans une maison n'est pas toujours propriétaire de cette maison, tandis que le propriétaire d'une maison ne doit pas nécessairement habiter dans celle-ci. Il incombe donc de représenter le fait de posséder une maison par une relation séparée et le fait d'habiter dans une maison par une relation séparée.

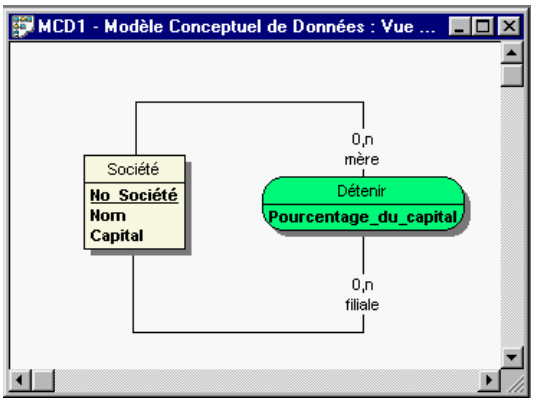
### *3.3.11.2 Relation réflexive et rôle d'une patte de relation*

Exemple 1:



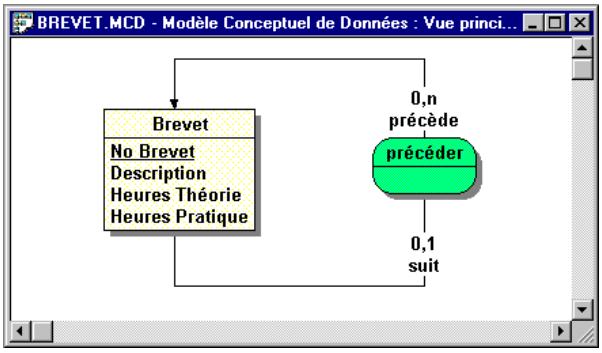

**Une relation réflexive, est une relation, dont les deux pattes sont liées à une même entité.** En général, la signification des pattes d'une relation réflexive devrait être clarifiée par l'indication d'un **rôle**.

Nous avons donc:




Exemple 2:

Afin d'obtenir une licence pour piloter un avion de ligne, un pilote doit effectuer un certain nombre de brevets. Il existe une hiérarchie prédéfinie en ce qui concerne les brevets (structure arborescente). A chaque fois qu'un pilote a réussi un brevet, il a la possibilité d'effectuer un certain nombre d'autres brevets, qui sont dépendants du brevet réussi. Tous les brevets sont dépendants du brevet de base.

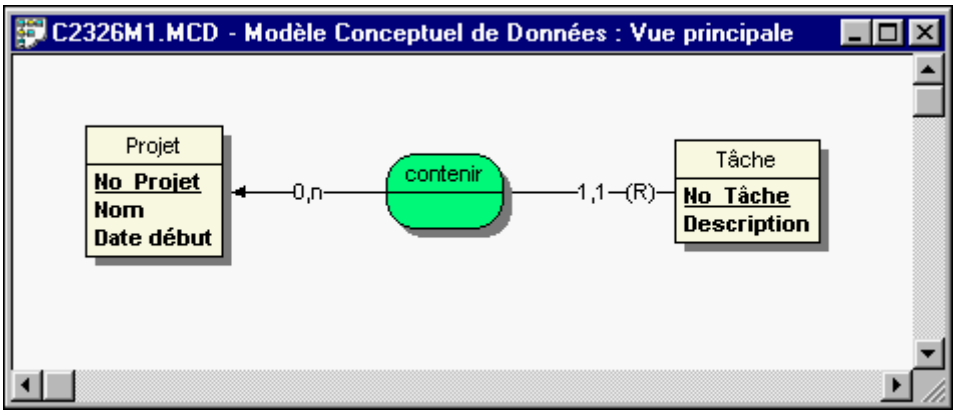



### *3.3.11.3 La notion d'identifiant relatif*

Sachant que chaque entité doit obligatoirement être dotée d'un identifiant, certaines entités ont cependant une existence complètement dépendante et liée à une autre entité. Une entité A est complètement dépendante d'une entité B, c.à.d. qu'une occurrence de l'entité A ne peut pas exister sans être reliée à une occurrence de l'entité B, lorsque les deux conditions suivantes sont vraies:

1. L'entité A est émettrice d'une CIF tandis que l'entité B est cible de la même CIF.
2. L'entité A n'est pas indépendante par rapport à la CIF (Cardinalité minimale = 1)

Exemple:



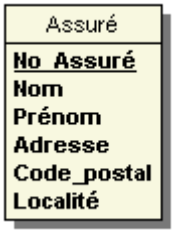

L'entité *Tâche* est complètement dépendante de l'entité *Projet*.

Dans le cas d'une telle dépendance complète, on peut avoir recours à un identifiant relatif. Dans notre exemple, la propriété **No_Tâche** constitue l'identifiant relatif de l'entité *Tâche*. Cette propriété ne remplit dans ce cas pas les conditions pour devenir identifiant absolu (Le même numéro de tâche est susceptible d'apparaître dans plusieurs projets). Toutefois, on peut affirmer qu'en relation à un certain numéro de projet, le numéro de tâche est un identifiant absolu.

On note cette identification relative par la lettre (R) sur la patte reliée à l'entité qui contient l'identifiant relatif.

### *3.3.11.4 Historisation*

Pour certaines propriétés, entités ou relations, on désire parfois conserver les valeurs antérieures en cas de modification. On parle dans ce contexte d'**historisation**. Théoriquement, cette idée n'est pas tout à fait en accord avec les règles de conception d'un système d'information. Prenons l'exemple suivant:




Pour une occurrence de cette entité, c.à.d. pour un assuré spécifique, il existe uniquement une seule valeur pour chaque propriété. Selon cette modélisation, un assuré ne peut par exemple pas habiter en même temps dans deux localités différentes. En général, ceci ne pose aucun problème, comme un assuré indique normalement une seule adresse de référence.

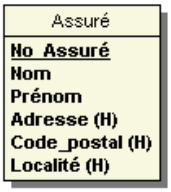
Toutefois, cette modélisation ne permet pas de représenter le tracé historique des adresses, lorsqu'un assuré déménage une ou plusieurs fois. Dans la plupart des cas, cette modélisation de l'historique n'est pas demandée, mais elle est quand même réalisable à l'aide de la méthode Merise. Au niveau conceptuel, nous indiquons simplement ce que nous voulons historiser.

Nous distinguons 3 cas:

### 1. Propriété historisée

La conservation des valeurs historiques s'applique à une ou plusieurs propriétés d'une entité ou d'une relation. Dans le MCD, on indique une historisation de propriété par la lettre (H) derrière la resp. les propriétés concernées.

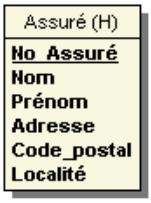
Exemple:




### 2. Entité historisée

La conservation des valeurs s'applique à toutes les propriétés d'une l'entité. On indique l'historisation par la lettre (H) derrière le nom de l'entité.

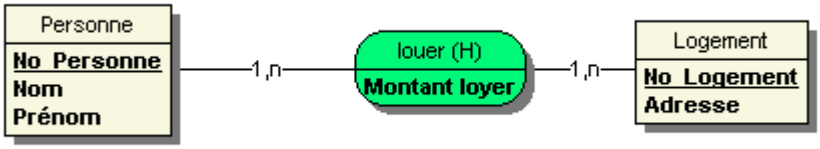
Exemple:




### 3. Relation historisée

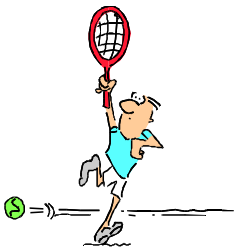
La conservation des valeurs s'applique à toutes les propriétés d'une relation. On indique l'historisation par la lettre (H) derrière le nom de la relation.

Exemple:




On ne peut pas historiser une relation sans propriétés, puisque l'expression 'historiser une relation' n'est qu'un abus de langage, il s'agit en fait d'historiser toutes les propriétés d'une relation. On peut remarquer à ce moment que la méthode MERISE présente une particularité en ce qu'elle ne prévoit pas l'historisation d'une propriété individuelle d'une relation

## 3.3.12 Exercices

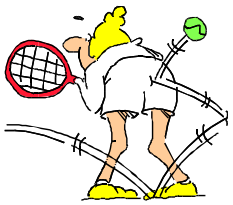


### Exercice 1

Un club de tennis vous demande d'informatiser la gestion des réservations des différents terrains. A ces fins, vous disposez des informations suivantes.

- Le club dispose d'une liste de membres. Quiconque veut jouer sur un des terrains, doit devenir membre du club.
- Un membre est caractérisé par un numéro interne au club, par son nom, prénom, adresse, code postal, localité, numéro de téléphone ainsi qu'une indication s'il est un joueur licencié auprès de la fédération de tennis ou non.
- Pour chaque réservation, on désire connaître l'identité des deux joueurs membres. Au cas où quatre joueurs réserveraient un terrain, uniquement deux joueurs sont enregistrés dans le système.
- Le club dispose de plusieurs terrains, dont certains sont couverts. On distingue en plus le type du terrain selon la nature du sol (p.ex. Sable, Herbe etc.)
- Une réservation se fait pour une date précise par tranches d'une heure.

Créez le MCD correspondant.



## Exercice 2

Une société aérienne utilise à présent les fiches suivantes pour la gestion des ressources.

**Vol No. : 98-8-798**

| | Date | Heure | Code Aéroport | Nom Aéroport | Ville | Pays |
|---|---|---|---|---|---|---|
| **Départ** | 24/08/98 | 7h45 | FIN | Findel | Lux | Lux |
| **Arrivée** | 24/08/98 | 9h00 | LHR | Heathrow | Lon | UK |

| **Avion** | | | | |
|---|---|---|---|---|
| **No** | **Marque** | **Type** | **Portée (km)** | **Capacité Passagers** |
| 23 | Boeing | 737-400 | 3810 | 147 |

| **Commandant** | | | | |
|---|---|---|---|---|
| **No** | **Nom** | **Prénom** | **Date de naissance** | **Brevet** |
| 726 | Weber | Jos | 13/06/65 | PP-IFR/EP/DA |

| **Co-pilote** | | | | |
|---|---|---|---|---|
| **No** | **Nom** | **Prénom** | **Date de naissance** | **Brevet** |
| 813 | Meier | Emil | 23/04/73 | PP-IFR |

| **Personnel de cabine** | | |
|---|---|---|
| **No** | **Nom** | **Prénom** |
| 1072 | Feller | Nathalie |
| 1014 | Pinto | Tania |
| 1103 | Weis | Laurent |
| | | |

Sachant que la société entretient déjà une BD avec tous les pilotes et qu'un pilote peut être commandant d'un vol et co-pilote d'un autre vol, proposez un MCD, qui permet l'informatisation de la gestion des ressources.

## Exercice 3

Un nouveau parc de vacances va prochainement ouvrir ses portes au Luxembourg. Dans ce parc, les visiteurs sont logés dans des bungalows. Vous êtes chargé de l'implémentation d'un système informatisé pour gérer les réservations des bungalows.

Après plusieurs réunions avec les responsables de la gestion du parc, vous avez collectionné les informations suivantes.

- Le parc est subdivisé en plusieurs zones, dont chacune contient environ 40 – 50 bungalows. Chaque zone dispose de ses propres magasins, restaurants, piscines etc. .

  Pour l'ouverture du parc, les zones suivantes sont prêtes à accueillir des visiteurs.

| Zone | Situation | Description |
|---|---|---|
| Texas | Nord | Imitation "Kloondike-City" avec Saloon, Sheriff Office . . . |
| Chine | Est | Chine traditionnelle avec temple, palais . . . |
| Hawaï | Sud-est | Atmosphère tropicale avec palmiers, mer artificielle . . . |
| Camelot | Sud | Ambiance médiévale autour d'un magnifique château ... |
| Liliput | Centre | Zone comportant plein d'éléments des contes bien connus |

  Les bungalows sont parfaitement intégrés dans l'atmosphère correspondante de leur zone.

- Chaque bungalow du parc appartient à une des catégories suivantes, de façon indépendante à sa situation (zone).

| Catégorie | Description | Capacité | Prix par nuit |
|---|---|---|---|
| A | Bain ou douche / WC sép. / TV | 3 | 1200 |
| B | Bain et douche / WC sép. / TV / Terrasse | 3 | 1500 |
| C | Bain ou douche / WC sép. / TV | 5 | 2000 |
| D | Bain et douche / WC sép. / TV / Terrasse | 5 | 2300 |
| E | Bain et douche / WC sép. / TV / Terrasse | 7 | 3000 |

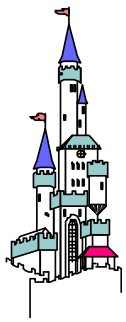
- Afin de faciliter la gestion des bungalows, le responsable du service Comptabilité vous demande de prévoir uniquement des nombres avec 2 positions pour numéroter les bungalows.

- Les clients peuvent effectuer des réservations. Une réservation concerne un seul bungalow. Suite à une réservation, une fiche de réservation est immédiatement envoyée au client. Deux semaines avant la date d'arrivée au parc, une facture correspondante est envoyée au client. Cette facture doit être réglée avant l'arrivée au parc. Le responsable de la facturation veut évidemment garder trace des informations contenues sur les factures. Le responsable de la réception désire voir dans le système si une facture correspondant à une réservation a déjà été payée ou non.

- Lors de la réservation d'un bungalow, le client a le choix entre les suppléments suivants.

| Code supplément | Description | Prix (par personne) par jour |
|---|---|---|
| 01 | Literie | 100 |
| 02 | Livraison à domicile du petit déjeuner | 300 |
| 03 | Livraison à domicile du quotidien | 50 |

- Voici un modèle d'une fiche de réservation




# Wonderland S.à r.l.

**Parc de bungalows**
3, am Boesch
L-8899 Schlindermanderscheid
G.D.Luxembourg
Tél: (Lux)+345566 / Fax: (Lux)+345567

## Fiche de réservation

| Client |
|---|
| Numéro: 340 |
| Nom: Weber |
| Prénom: Jos |
| Adresse: 23, rue Principale |
| Code postal: L-8765 |
| Localité: Grevenmacher |
| Pays: Luxembourg |
| No. Passeport: 87699 |
| No. Téléphone: (Lux)+348845 |

| Réservation |
|---|
| No: 589 |
| Date d'arrivée: 03/09/98 |
| Date de départ: 07/09/98 |
| Nombre de personnes: 4 |
| **Bungalow** |
| Zone: Liliput |
| Numéro: 19 |
| Catégorie: Bain et douche / WC sép. / TV / Terrasse |
| Capacité: 5 |

| Suppléments | |
|---|---|
| Code supplément | Description |
| 01 | Literie |
| 03 | Livraison à domicile du quotidien |
| | |

Une facture vous sera envoyée environ 2 semaines avant votre arrivée au parc. Cette facture est à régler avant l'arrivée au parc. Nous vous souhaitons un beau séjour au parc Wonderland. Si vous avez encore des questions, n'hésitez pas à nous contacter.

*Arsène Lupin*

RESERVATION MANAGER

- Une facture reprend exactement les mêmes informations, avec en plus la date d'envoi de la facture et le prix total à payer.

- Afin d'établir des statistiques, la direction du parc est intéressée de sauvegarder dans le système l'évolution des prix par nuit pour les différentes catégories de bungalows.

- Un client est uniquement considéré comme tel à partir de la première fois qu'il effectue une réservation.

Créez le modèle conceptuel

# 3.4 Le modèle logique des données (MLD)

## 3.4.1 Définition

Jusqu'à présent nous avons établi des MCD basés sur une analyse d'un domaine bien défini (p.ex. Gestion des séances d'un cinéma, Gestion des séjours des patients d'un hôpital etc.). La finalité d'un MCD est de nous faciliter la création d'une base de données pour gérer un tel domaine.

Nous savons également qu'une base de données est constituée par un ensemble de tables, dont chacune est composée de champs de données.

Hors le MCD ne connaît pas la notion de table, tandis qu'une base de données ne connaît pas le concept des entités reliées entre-elles via des relations portant des cardinalités.

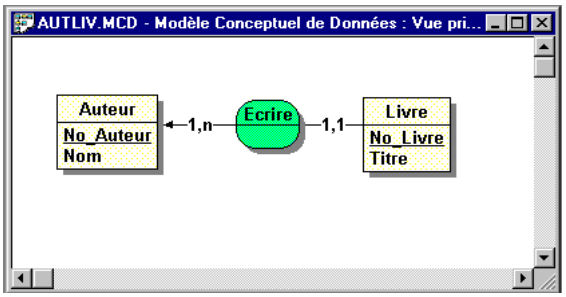
Pour cela, il existe un autre modèle, le **modèle logique des données (MLD)**, qui utilise essentiellement le formalisme des tables logiques. Un MLD, qui est toujours basé sur un MCD donné, contient donc toutes les informations de ce MCD, mais les représente à l'aide d'un formalisme différent qui est très adapté aux structures d'une base de données.

Tandis que le MCD représente un système d'information d'une façon générale et indépendante d'un système informatique, le MLD tient compte de la réalisation par le biais d'un SGBD.

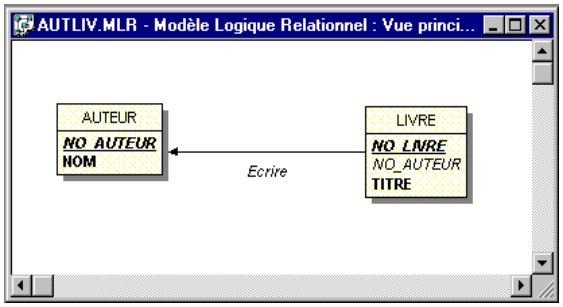
**Un MLD est essentiellement composé de tables logiques reliées entre elles par des flèches.**

Voici un exemple qui montre un MCD avec son MLD correspondant:

MCD




MLD



## Exercice

En vous référant à l'exemple précédant, répondez brièvement aux questions suivantes.

1. Comment est-ce qu'on traduit une entité du MCD dans le MLD ?

2. Comment est-ce qu'on traduit une propriété d'une entité du MCD dans le MLD ?

3. Comment est-ce qu'on traduit un identifiant d'une entité du MCD dans le MLD ?

4. Comment est-ce qu'on traduit la relation *Ecrire* avec ses cardinalités du MCD dans le MLD ?

5. Le MCD nous dit que chaque livre est uniquement écrit par un seul auteur (cardinalité max.), tandis qu'un auteur peut écrire plusieurs livres. Comment est-ce qu'on peut retrouver ces informations dans le MLD ?

Remarque:

La méthode MERISE définit de façon générale certaines règles qui nous permettront de transformer n'importe quel MCD en MLD.

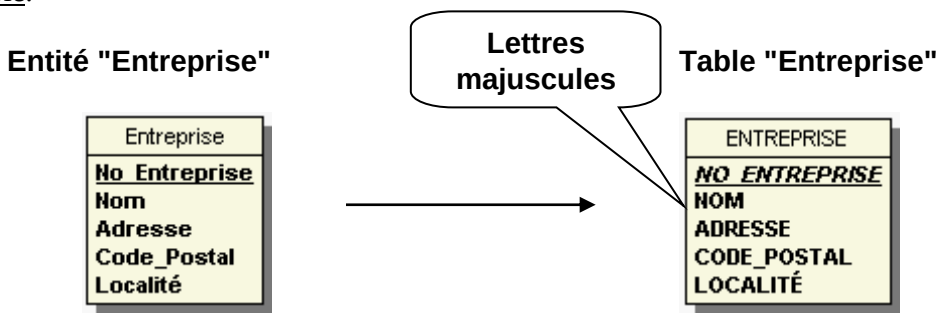
## 3.4.2 Règles de transformation du MCD au MLD

Nous allons définir les règles de transformation pour le passage du MCD au MLD, en respectant les différents cas qui se posent.

### *3.4.2.1 Transformation des entités*

**Toute entité est transformée en table. Les propriétés de l'entité deviennent les attributs de la table. L'identifiant de l'entité devient la clé primaire de la table.**

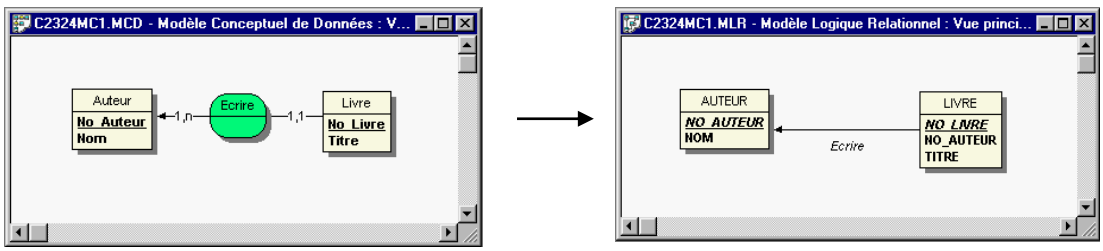
Exemple:




### *3.4.2.2 Transformation des relations binaires du type[1] (x,n) – (x,1)*

**Afin de représenter la relation, on duplique la clé primaire de la table basée sur l'entité à cardinalité (x,n) dans la table basée sur l'entité à cardinalité (x,1). Cet attribut est appelé clé étrangère. Les deux tables sont liées par une flèche nommée selon la relation, qui pointe de la table à clé étrangère vers la table qui contient la clé primaire correspondante.**

Exemple:




L'attribut *No_Auteur* qui est clé primaire de la table *Auteur*, devient clé étrangère dans la table *Livre*.

[1] x peut prendre les valeurs 0 ou 1

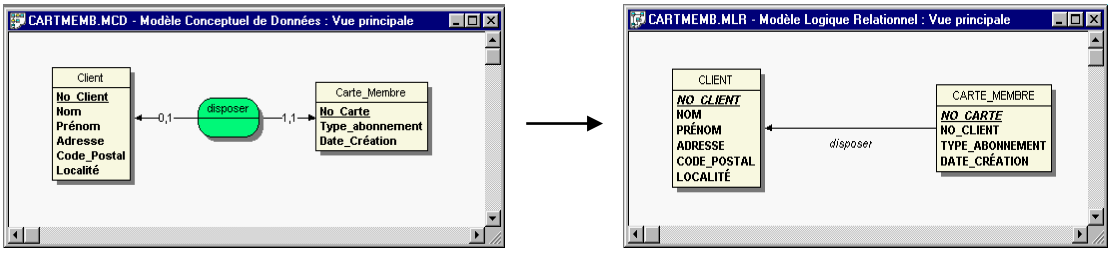
### 3.4.2.3 Transformation des relations binaires du type (x,1) – (x,1)

Nous devons distinguer plusieurs cas. Sachant qu'une relation binaire du type (1,1)-(1,1) ne doit pas exister il nous reste les 2 cas suivants:

**Relation binaire (0,1)-(1,1)**

**⚠ On duplique la clé de la table basée sur l'entité à cardinalité (0,1) dans la table basée sur l'entité à cardinalité (1,1).**

Exemple:

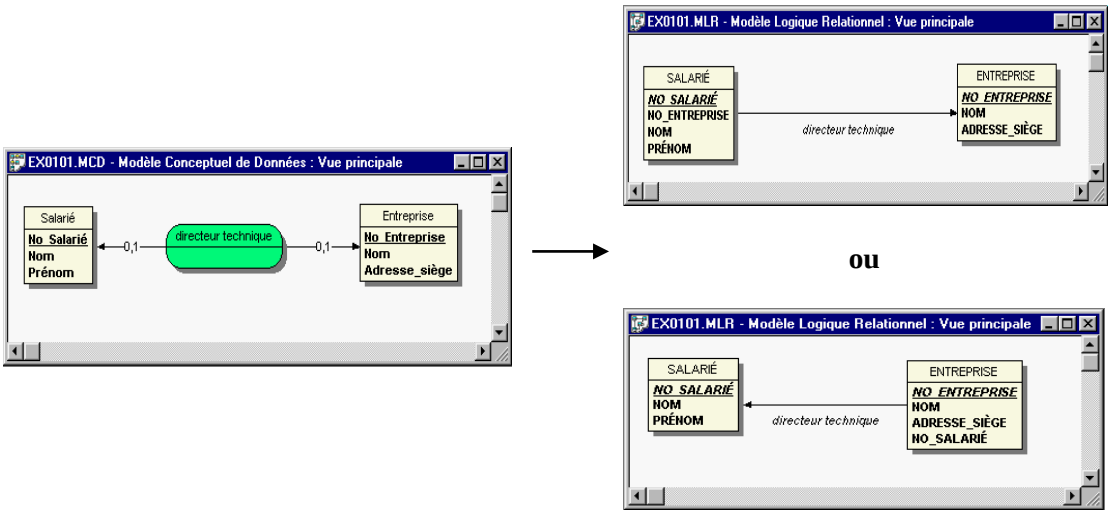



Le *No_Client*, qui est clé primaire de la table *Client*, devient clé étrangère dans la table *Carte_Membre*.

**Relation binaire (0,1)-(0,1)** !!! Ne figure actuellement pas au programme de la classe 13CG

**⚠ On duplique la clé d'une des tables dans l'autre. Lorsque la relation contient elle-même des propriétés, celles-ci deviennent également attributs de la table dans laquelle a été ajoutée la clé étrangère.**

Exemple:

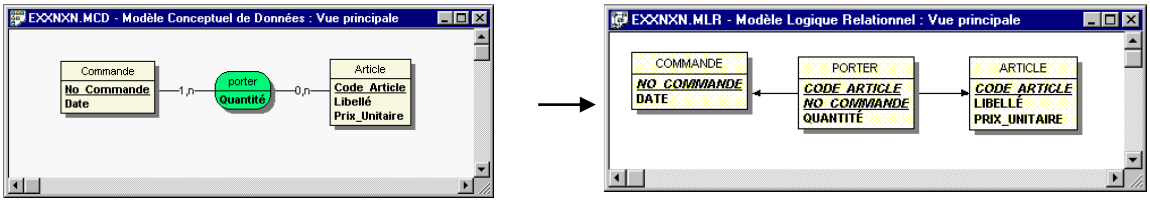



Soit on migre la clé primaire de la table *Entreprise* dans la table *Salarié*, soit on fait l'inverse.

### 3.4.2.4 Transformation des relations binaires du type (x,n) – (x,n)

**On crée une table supplémentaire ayant comme clé primaire une clé composée des clés primaires des 2 tables. Lorsque la relation contient elle-même des propriétés, celles-ci deviennent attributs de la table supplémentaire. Une propriété de la relation qui est soulignée devra appartenir à la clé primaire composée de la table supplémentaire.**

Exemple:

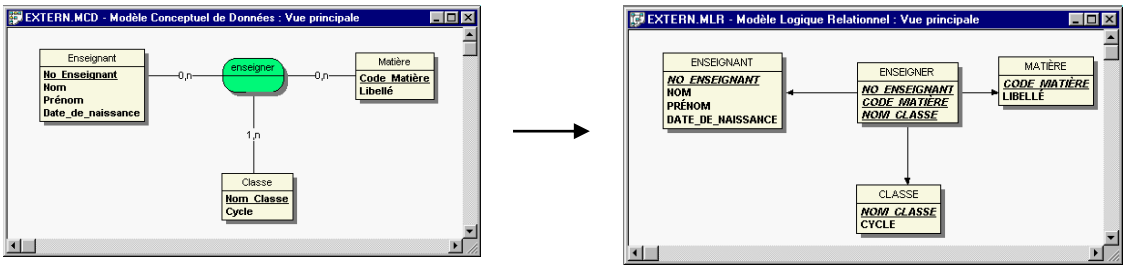



On crée une table *Porter*, qui contient comme clé primaire une clé composée *de No-Commande* et *Code_Article*. Elle contient également la propriété *Quantité* issue de la relation *Porter*.

### 3.4.2.5 Transformation des relations ternaires

**On crée une table supplémentaire ayant comme clé primaire une clé composée des clés primaires de toutes les tables reliées. Cette règle s'applique de façon indépendante des différentes cardinalités. Lorsque la relation contient elle-même des propriétés, celles-ci deviennent attributs de la table supplémentaire. Une propriété de la relation qui est soulignée devra appartenir à la clé primaire composée de la table supplémentaire.**

Exemple:

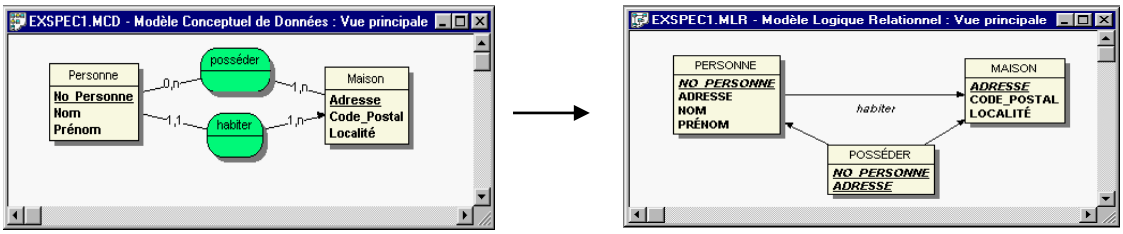



La table *Enseigner* contient une clé composée de *No_Enseignant*, *Code_Matière* et *Nom_Classe*.

### 3.4.2.6 Transformation de plusieurs relations entre 2 entités

**Les règles générales s'appliquent**

Exemple:

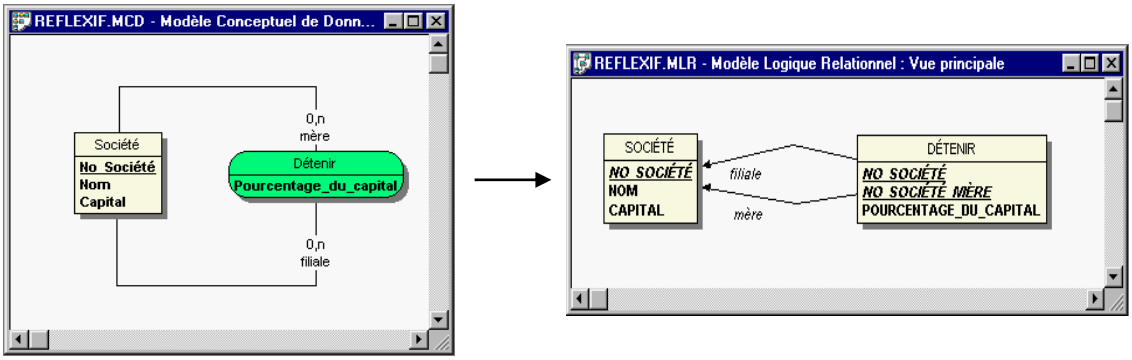



La relation *habiter* du type (x,n)-(x,1), est traduite par la migration de l'attribut *Adresse* dans la table *Personne*. La relation *posséder* du type (x,n)-(x,n) est traduite par la création d'une table supplémentaire du même nom. Cette table contient comme clé primaire composée, les clés des deux tables reliées *Personne* et *Maison*. On a donc simplement appliqué 2 fois de façon indépendante les règles de transfert MCD → MLD.

### 3.4.2.7 Transformation des relations réflexives

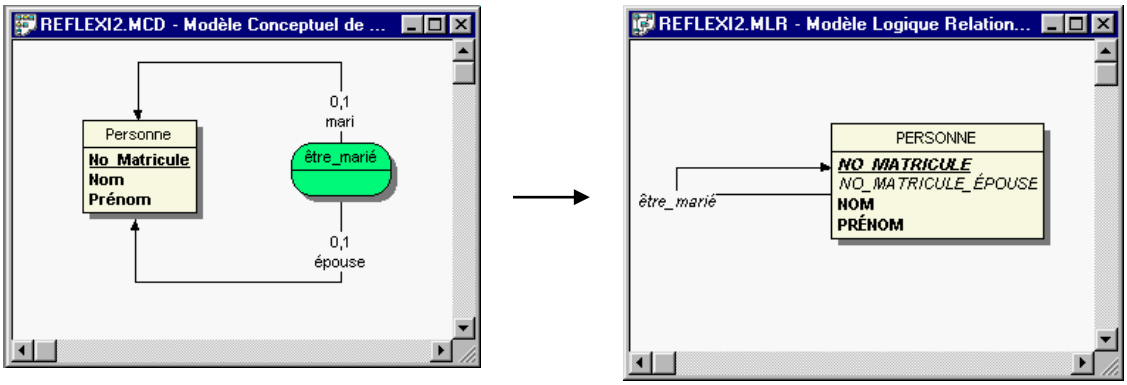
**Nous appliquons les règles générales avec la seule différence que la relation est 2 fois reliée à la même entité**

Exemple 1:




Comme il s'agit d'une relation (x,n)-(x,n), une table supplémentaire est créée. Cette table contient comme clé primaire composée, la clé des "deux" entités reliées. Comme la même entité est liée 2 fois à la relation, on ne peut pas utiliser 2 fois le même nom pour la clé. Dans ce cas il convient d'utiliser des rôles dans le MCD, et d'intégrer le rôle dans le nom d'une des clés migrées dans le MLD.

Exemple 2:

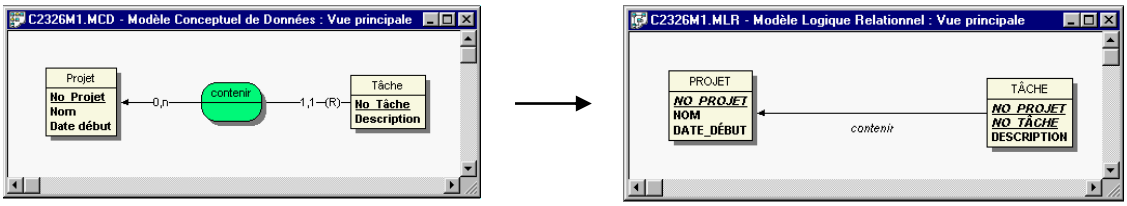



Comme il s'agit d'une relation (0,1)-(0,1), nous avons en général le choix en ce qui concerne quelle entité contiendra la clé étrangère. Comme cette relation est liée deux fois à la même entité, il est évident que nous devons dupliquer la clé primaire, tout en veillant que le même nom de clé ne sera pas utilisé pour la clé primaire et la clé étrangère. Dans notre exemple, tous les hommes mariés, ont comme valeur de la clé étrangère la matricule de leur épouse actuelle. Pour les hommes non mariés et les femmes, la clé étrangère est sans valeur. On pourrait bien sûr utiliser la modélisation inverse avec une clé étrangère *NO_MATRICULE_MARI*, qui indique pour chaque femme mariée, la matricule de son mari.

### *3.4.2.8 Transformation de l'identifiant relatif*

**Sachant que l'entité dépendante est toujours liée à la relation par les cardinalités (1,1), nous pouvons appliquer les règles générales. Dans chaque cas, la table issue de l'entité dépendante contient donc comme clé étrangère, la clé primaire de l'autre table. L'identification relative est représentée par le fait que la table issue de l'entité dépendante contient une clé primaire composée, constituée de la clé primaire transformée de l'identifiant de cette entité et de la clé étrangère.**

Exemple:



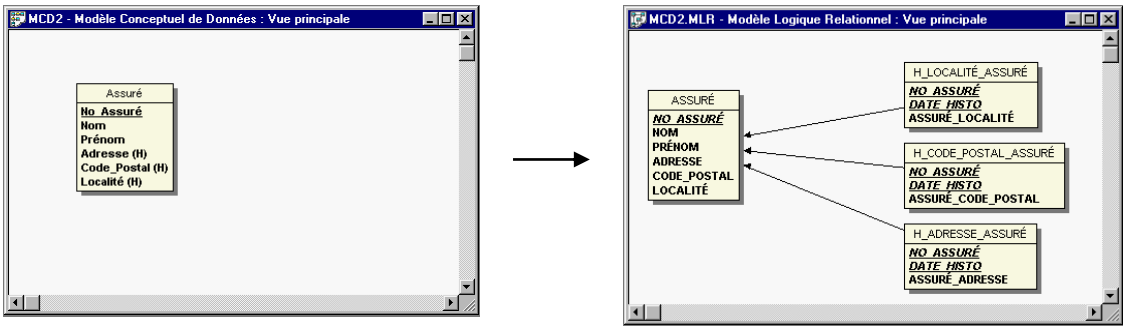

Tout en respectant les règles générales du passage MCD→MLD, la clé primaire de la table *Projet* migre comme clé étrangère dans la table *Tâche*. L'identification relative est représentée par le fait que la table tâche contient une clé primaire composée de *No_Tache* et *No_Projet*.

### 3.4.2.9 Transformation de l'historisation

**1. Historisation d'une propriété**

**Pour chaque propriété à historiser, on crée une table qui contient:**
- **Une clé primaire composée de la clé primaire de la table qui contient la propriété à historiser et de la date d'historisation.**
- **La propriété à historiser**

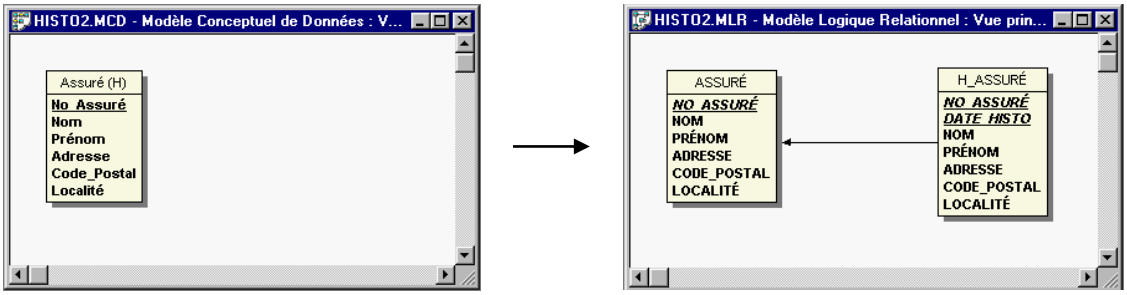
Exemple:




**2. Historisation d'une entité**

**Pour toute modification de valeur d'une des propriétés de l'entité, on historise l'ensemble des valeurs des propriétés dans une table qui contient:**
- **Une clé primaire composée de la clé primaire de l'entité à historiser et de la date d'historisation**
- **Toutes les autres propriétés de l'entité à historiser**

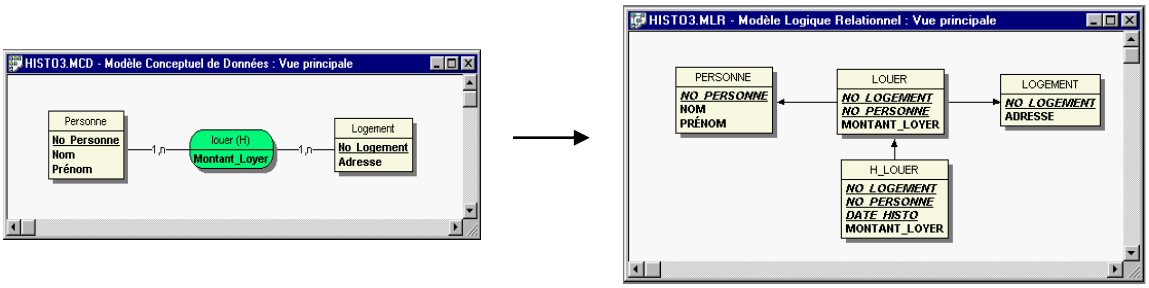
Exemple:




**3. Historisation d'une relation**

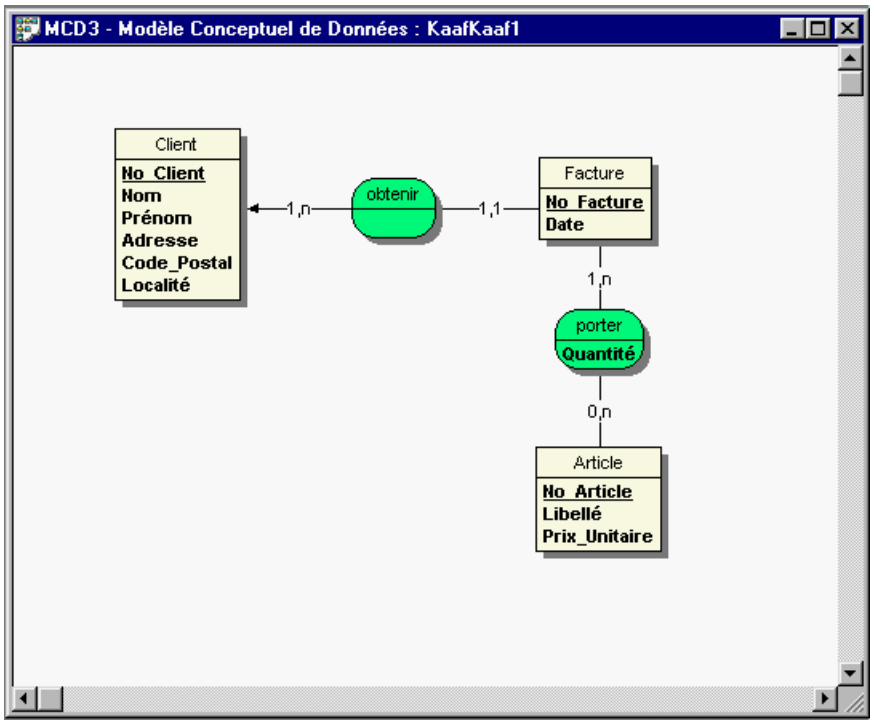
**Pour toute modification de valeur d'une des propriétés de la relation, on historise l'ensemble des valeurs des propriétés dans une table qui contient:**
- **Une clé primaire composée de la clé primaire de la table qui représente la relation à historiser et de la date d'historisation**
- **Toutes les autres propriétés de la relation à historiser**

Exemple:

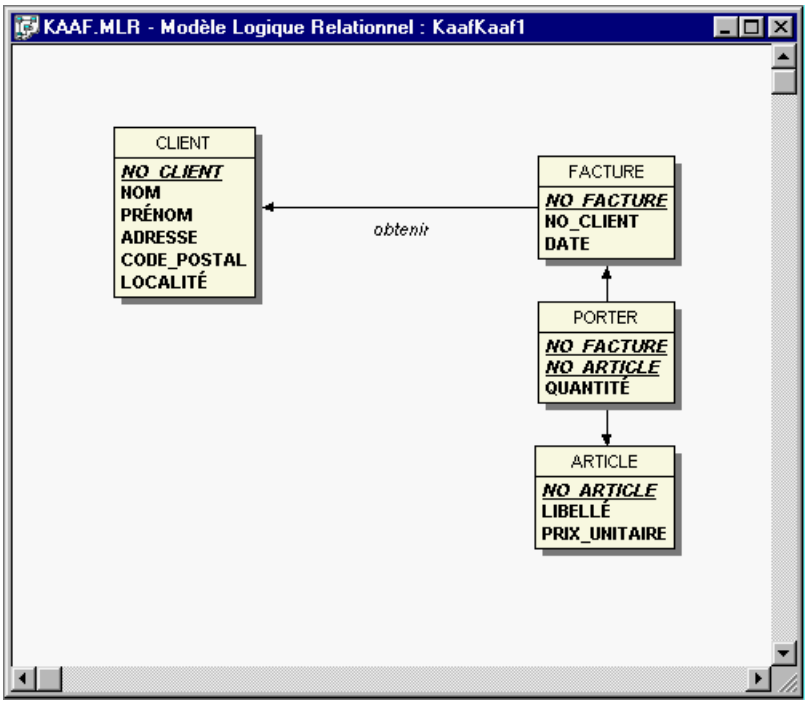

HISTO3.MCD - Modèle Conceptuel de Données : Vue principale
Personne
No_Personne
Nom
Prénom
louer (H)
Montant_Loyer
1,n
1,n
Logement
No_Logement
Adresse
HISTO3.MLR - Modèle Logique Relationnel : Vue principale
PERSONNE
NO_PERSONNE
NOM
PRÉNOM
LOUER
NO_LOGEMENT
NO_PERSONNE
MONTANT_LOYER
LOGEMENT
NO_LOGEMENT
ADRESSE
H_LOUER
NO_LOGEMENT
NO_PERSONNE
DATE_HISTO
MONTANT_LOYER

### 3.4.3 Exemple "KaafKaaf"

Transformez le MCD suivant, qui représente la facturation de la société "KaafKaaf" (voir chapitre 3.3.6 Exemple "KaafKaaf"), en un MLD en respectant toutes les règles du passage MCD → MLD.




Voici une solution:



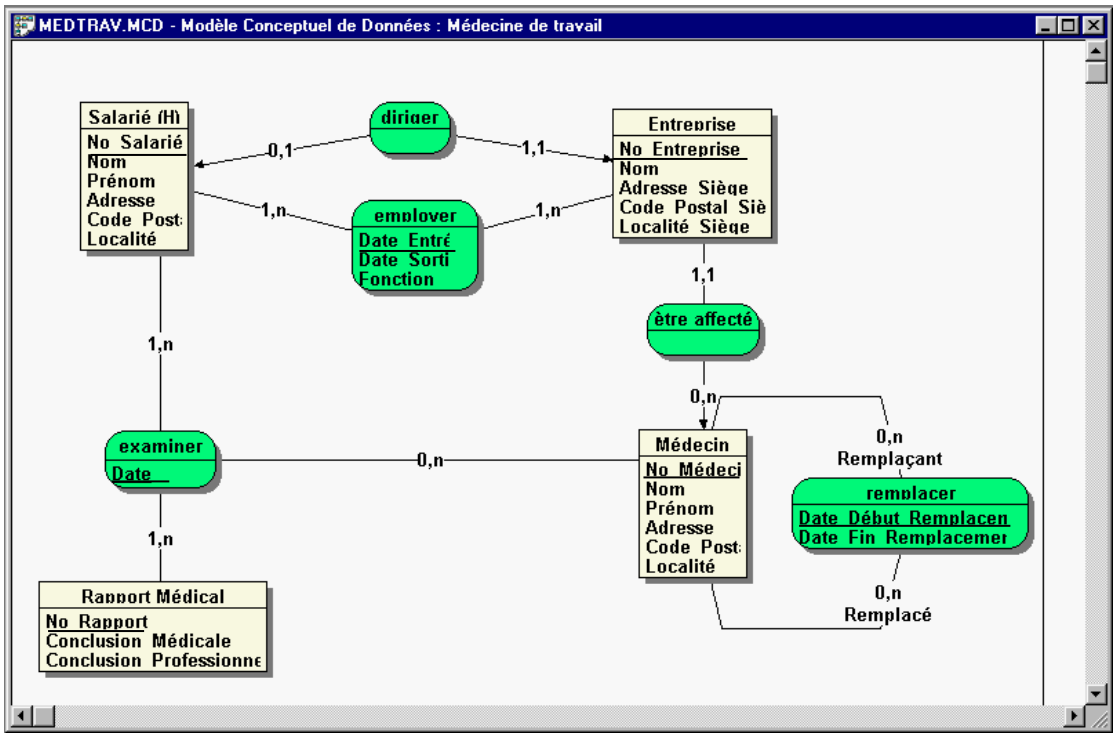
### 3.4.4 Exercices

**Exercice 1**

Transformez les MCD que vous avez réalisés pour les exercices 1 à 6 du chapitre 3.3.10 et les exercices 1 à 3 du chapitre 3.3.12 en MLD.

## Exercice 2

Transformez le MCD suivant en MLD en respectant toutes les règles de passage MCD→MLD.

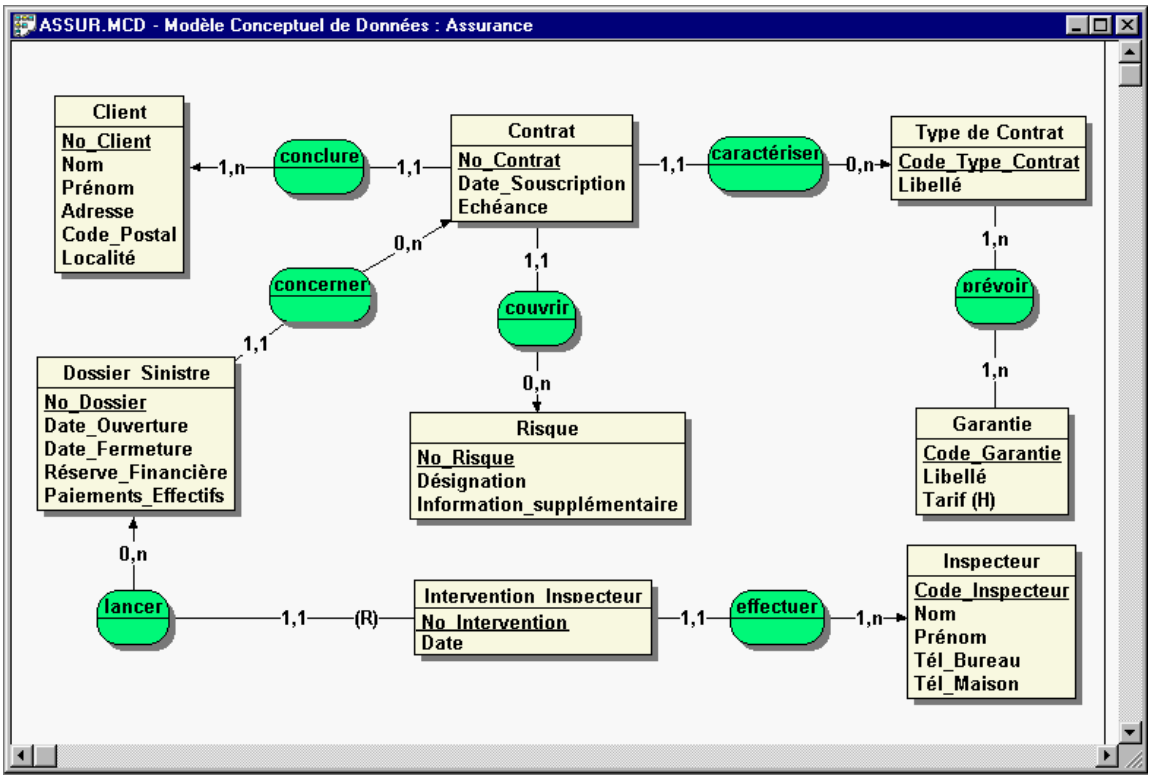



Remarques:

- En ce qui concerne le rapport médical, une conclusion médicale pourrait par exemple être "Infection" ou "Cancer de la gorge", tandis que la conclusion professionnelle qui s'en suit serait par exemple "Apte" ou "Inaptitude temporaire <x> jours".
- L'entité *Salarié* est historisée.

## Exercice 3

Voici un MCD qui représente de façon très simplifiée la gestion d'une compagnie d'assurances. Transformez le MCD en MLD en respectant toutes les règles de passage MCD→MLD.




Remarques:

- Le type de contrat indique les garanties prévues.
  Exemple: Type AUTO-SIMPLE contient (RC-AUTO et Protection juridique)
  Type AUTO-SPECIAL contient (Garanties AUTO-SIMPLE + FEU + VOL)
  Type AUTO-DELUXE contient (Garanties AUTO-SPECIAL + Dégâts matériels)
- Un contrat couvre un seul risque. Ce risque peut être une voiture ou une habitation.
- Afin d'améliorer l'exploitation statistique des données concernant les garanties, le tarif d'une garantie est historisé.

# 3.5 Le modèle physique des données (MPD)

## 3.5.1 Définition

Le modèle physique des données (MPD) est la traduction du modèle logique des données (MLD) dans une structure de données spécifique au système de gestion de bases de données (SGBD) utilisé.

Le MPD est donc représenté par des tables définies au niveau du système de gestion de bases de données. C'est donc au niveau du MPD que nous **quittons la méthode générale** de création d'un MCD et de sa transformation en MLD, pour nous tourner vers la **manipulation d'un SGBD spécifique**.

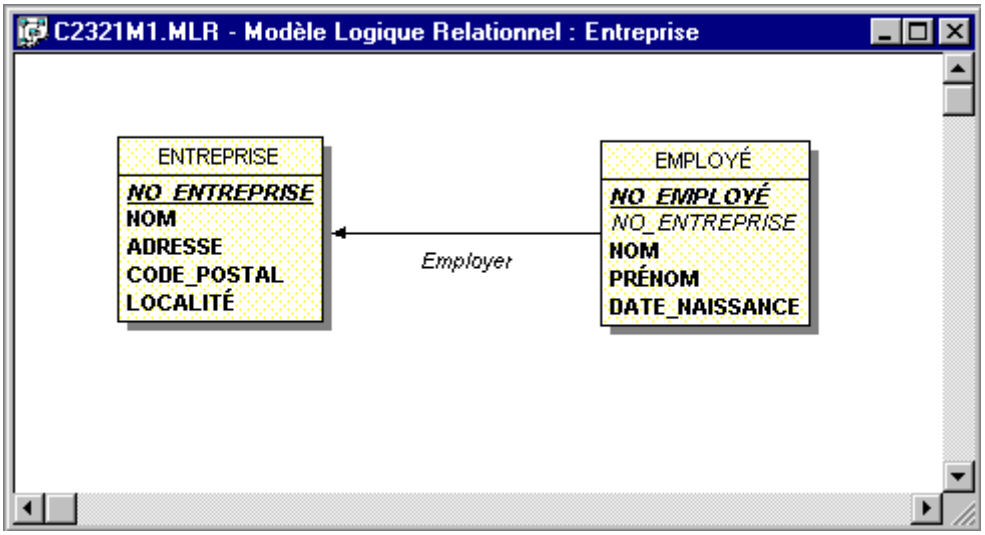
## 3.5.2 Passage du MLD au MPD

**Le passage MLD → MPD se fait par les étapes suivantes:**

- **Implémentation physique de chaque table du MLD dans le SGBD utilisé.**
- **Pour chaque table, indiquer au SGBD quel(s) champ(s) constitue(nt) la clé primaire.**
- **Pour chaque table, indiquer au SGBD la (les) clé(s) étrangère(s), et la (les) clé(s) primaire(s) correspondante(s).**

Pour ce faire, la plupart des SGBD actuellement sur le marché nous offrent 2 possibilités.

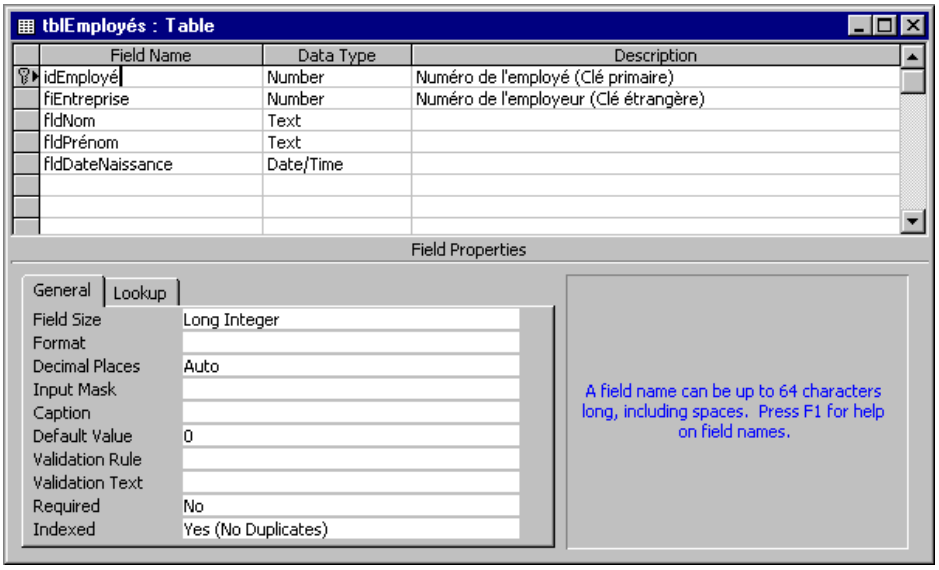
Prenons à titre d'exemple l'implémentation du modèle logique suivant.



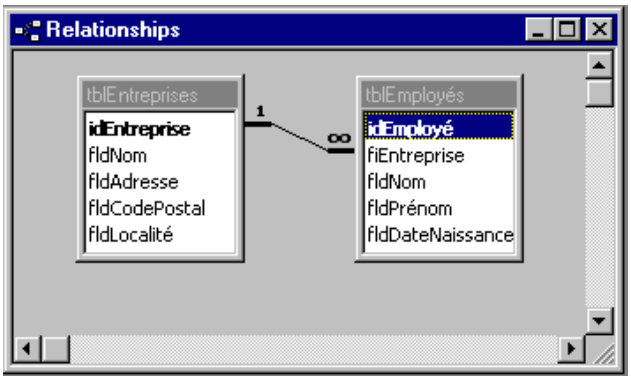
1. **Utilisation d'une ou de plusieurs interfaces graphiques, qui nous aident dans la création des tables physiques, dans la définition des clés primaires et dans la définition des relations.**

Exemple:

Définition de la table des employés avec le champ *idEmployé* étant défini comme clé primaire.




Définition de la relation entre les deux tables.




Remarquez que les noms des différents champs ont été modifiés lors de l'implémentation du modèle logique. Cette mesure dépend uniquement de la convention des noms utilisée et n'affecte pas du tout le fonctionnement correcte de la BD.

## 2. Utilisation de commandes spéciales, faisant partie d'un langage de définition de données (p.ex. SQL-DDL)

Exemple:

Implémentation du même modèle logique à l'aide de commandes spécifiques

```
REM ================================================================================
REM                Génération d'une base de données
REM                         SQL Générique (SQL 2)
REM                           (6/9/1998 17:03:24)
REM ================================================================================
REM      Nom de la base : Entreprises
REM      Projet :
REM      Auteur : Pierre Stockreiser
REM      Date de dernière modification : 6/9/1998 17:03:13
REM ================================================================================
REM ================================================================================
REM       TABLE : tblEntreprises
REM ================================================================================

CREATE TABLE tblEntreprises
   (
   idEntreprise INTEGER NOT NULL ,
   fldNom CHAR (20) NOT NULL ,
   fldAdresse CHAR (25) NOT NULL ,
   fldCodePostal CHAR (7) NOT NULL ,
   fldLocalité CHAR (20) NOT NULL ,

     PRIMARY KEY (idEntreprise) CONSTRAINT PK_ENTREPRISE
   );

REM ================================================================================
REM       INDEX DE LA TABLE tblEntreprises
REM ================================================================================

CREATE UNIQUE INDEX I_PK_ENTREPRISE
     ON tblEntreprises (idEntreprise ASC);

REM ================================================================================
REM       TABLE : tblEmployes
REM ================================================================================

CREATE TABLE tblEmployes
   (
   idEmploye INTEGER NOT NULL ,
   fiEntreprise INTEGER NOT NULL ,
   fldNom CHAR (32) NOT NULL ,
   fldPrénom CHAR (32) NOT NULL ,
   fldDateNaissance DATE NOT NULL ,

     PRIMARY KEY (idEmploye) CONSTRAINT PK_EMPLOYÉ
   );

REM ================================================================================
REM       INDEX DE LA TABLE tblEmployes
REM ================================================================================

CREATE UNIQUE INDEX I_PK_EMPLOYÉ
     ON tblEmployes (idEmploye ASC);

CREATE  INDEX I_FK_EMPLOYER
     ON tblEmployes (fiEntreprise ASC);


REM ================================================================================
REM       CREATION DES REFERENCES DE TABLE
REM ================================================================================


ALTER TABLE tblEmployes ADD (FOREIGN KEY (fiEntreprise)
      REFERENCES tblEntreprises (idEntreprise)
      CONSTRAINT FK_EMPLOYER
```

```
      ON UPDATE RESTRICT
      ON DELETE RESTRICT);

REM ------------------------------------------------------------------------
REM                 FIN DE GENERATION
REM ------------------------------------------------------------------------
```

Que vous avez utilisé l'une ou l'autre des 2 méthodes, le résultat sera toujours un ensemble de tables physiques reliées entre elles, dans lesquelles vous pouvez stocker des données.

## 3.5.3 Les contraintes d'intégrité

Une modélisation correcte et cohérente est sans doute une condition nécessaire pour créer une BD fonctionnelle, mais ne vaut pas grand chose, lorsque le SGBD utilisé pour implémenter la base, ne garantit pas l'intégrité de celle-ci lors du travail journalier avec les données.

Exemples:

- Le système doit empêcher un utilisateur à entrer une valeur double ou indéterminée (NULL) pour un champ déclaré comme clé primaire.
- Le système doit vérifier qu'une quantité livrée est toujours inférieure ou égale à une quantité commandée.

### *3.5.3.1 Les types de contraintes d'intégrité*

La **contrainte d'intégrité des tables** (angl. Table Integrity Constraint)

Cette contrainte vérifie qu'il n'existe jamais des doublons ou des valeurs indéterminées pour le(s) champ(s) qui constitue(nt) la clé primaire. La **clé primaire** doit donc toujours être unique et bien définie.

Exemples de violation de cette contrainte

A. L'ajout d'une valeur de clé primaire qui existe déjà dans la table.
B. La modification d'une valeur de clé primaire vers une valeur qui existe déjà dans la table.
C. L'ajout d'une valeur indéterminée pour une clé primaire.
D. La modification d'une valeur de clé primaire vers une valeur indéterminée.

Méthodes pour vérifier cette contrainte d'intégrité dans un SGBD

Dans un SGBD il suffit généralement de déclarer un ou plusieurs champs comme clé primaire d'une table pour que cette contrainte soit automatiquement vérifiée pour chaque insertion ou modification d'une valeur dans la table.

La **contrainte d'intégrité référentielle** (angl. Referential Integrity Constraint)

Par contrainte d'intégrité référentielle, on entend l'obligation qu'à chaque valeur d'une **clé étrangère** correspond une et une seule valeur de la clé primaire associée. Cette obligation doit toujours être vérifiée lors de l'ajout, de la suppression ou de la modification de données.

Exemples de violation de cette contrainte

A. L'ajout d'une clé étrangère pour laquelle il n'existe pas de valeur correspondante dans la clé primaire associée.
B. La modification d'une clé étrangère vers une valeur pour laquelle il n'existe pas de valeur correspondante dans la clé primaire associée.

C. La suppression d'une clé primaire, qui est référencée par une ou plusieurs valeurs d'une clé étrangère.
D. La modification d'une clé primaire, qui est référencée par une ou plusieurs valeurs d'une clé étrangère.

Méthodes pour vérifier cette contrainte d'intégrité dans un SGBD

Un SGBD nous offre généralement une ou plusieurs des quatre méthodes suivantes pour spécifier à tout moment l'intégrité référentielle des données d'une BD.

I. **Interdiction** des opérations du type A, B, C et D. Bien que cette possibilité soit très efficace, il existe parfois une alternative préférable en fonction de la nature des données.
   Remarque: Tandis que les opérations A et B ne sont jamais permises lorsque l'intégrité référentielle est appliquée, il n'en est pas de même pour les opérations C et D. Le langage SQL par exemple contient une commande `CREATE TABLE` qui permet de définir la structure d'une table. Cette commande contient entre autres les options `ON DELETE NO ACTION` et `ON UPDATE NO ACTION` qui permettent de réaliser explicitement l'interdiction des opérations C et D.

II. **Cascade** des opérations du type C et D vers les clés étrangères correspondantes. Une modification d'une clé primaire aurait comme conséquence la modification de toutes les clés étrangères correspondantes. Une suppression d'une clé primaire aurait comme conséquence la suppression automatique de tous les enregistrements dont la clé étrangère a la même valeur. Cette option est à utiliser avec précaution !!!
   Remarque: En SQL la commande `CREATE TABLE` nous offre les options `ON DELETE CASCADE` et `ON UPDATE CASCADE`.

III. Affectation d'une **valeur par défaut** aux clés étrangères concernées par une opération du type C ou D.

IV. Affectation d'une **valeur indéterminée** (NULL) aux clés étrangères concernées par une opération du type C ou D.

## La **contrainte d'intégrité générale** (angl. General Integrity Constraint)

Une contrainte d'intégrité générale est utilisée pour limiter les valeurs possibles d'un champ quelconque d'une table en fonction de la nature de celui-ci et de son rôle dans le système d'information.

Exemples de violation de cette contrainte

Il existe plusieurs variantes de cette contrainte.

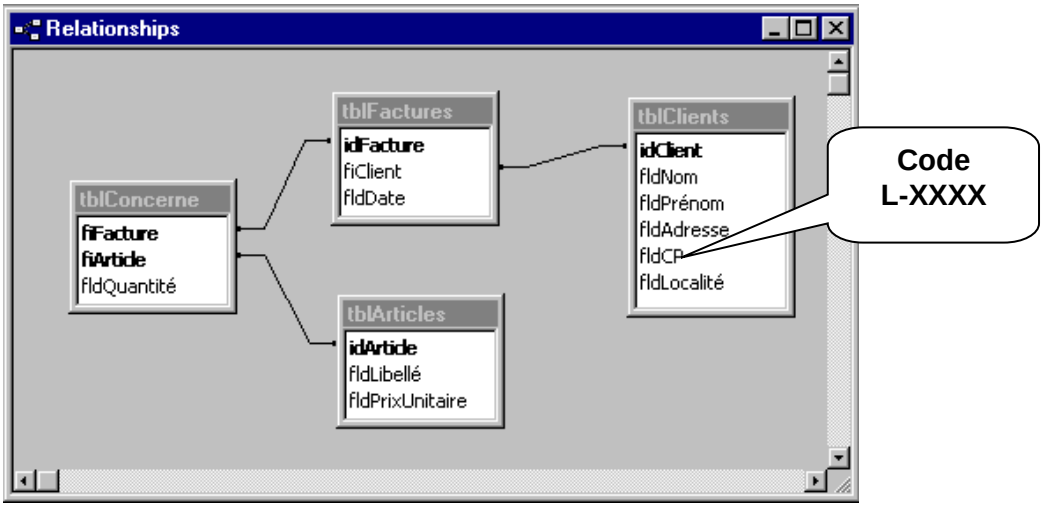
1. A chaque champ correspond un **type de données**, une **longueur** et un **format** bien définis.
   Exemples: Le numéro client doit être une valeur numérique.
   Le nom du client ne doit pas dépasser 25 caractères.
   Un numéro de compte doit respecter le format X-XXX/XX-X

2. Un champ peut avoir un **domaine de valeurs prédéfini** (une plage de valeurs possibles) et/ou une **valeur par défaut**.
   Exemples: Une note d'un devoir en classe doit être entre 0 et 60
   La prix d'une facture ne doit pas être un nombre négatif.
   La date d'une commande doit automatiquement être la date actuelle à moins que l'utilisateur n'entre une autre date.

3. La **valeur d'un champ peut limiter les valeurs possibles pour un autre champ** d'une table/d'une BD.
   Exemple: La valeur du champ *fldDatePaiement* est supérieure ou égale à la valeur du champ *fldDateFacture* pour une table *tblFactures*.

## Exercice

Soit la BD suivante.




a) Quelle(s) contrainte(s) est(sont) concernée(s) lors de l'ajout d'un client ?

b) Quelle(s) contrainte(s) est(sont) concernée(s) lors de la suppression d'un client ?

c) Quelle(s) contrainte(s) est(sont) concernée(s) lors de l'ajout d'une facture ?

d) Quelle(s) contrainte(s) est(sont) concernée(s) lors de la suppression d'une facture ?

e) Quelle(s) contrainte(s) est(sont) concernée(s) lors de l'ajout d'un enregistrement dans la table *tblConcerne* ?

f) Quelle(s) contrainte(s) est(sont) concernée(s) lors de la suppression d'un enregistrement de la table *tblConcerne* ?

g) Quelle(s) contrainte(s) est(sont) concernée(s) lors de la modification du numéro d'un article dans la table des articles?

h) Quelle(s) contrainte(s) est(sont) concernée(s) lors de la modification du numéro client d'une facture ?

# 4. Utilisation d'un outil de modélisation

## 4.1 Définition

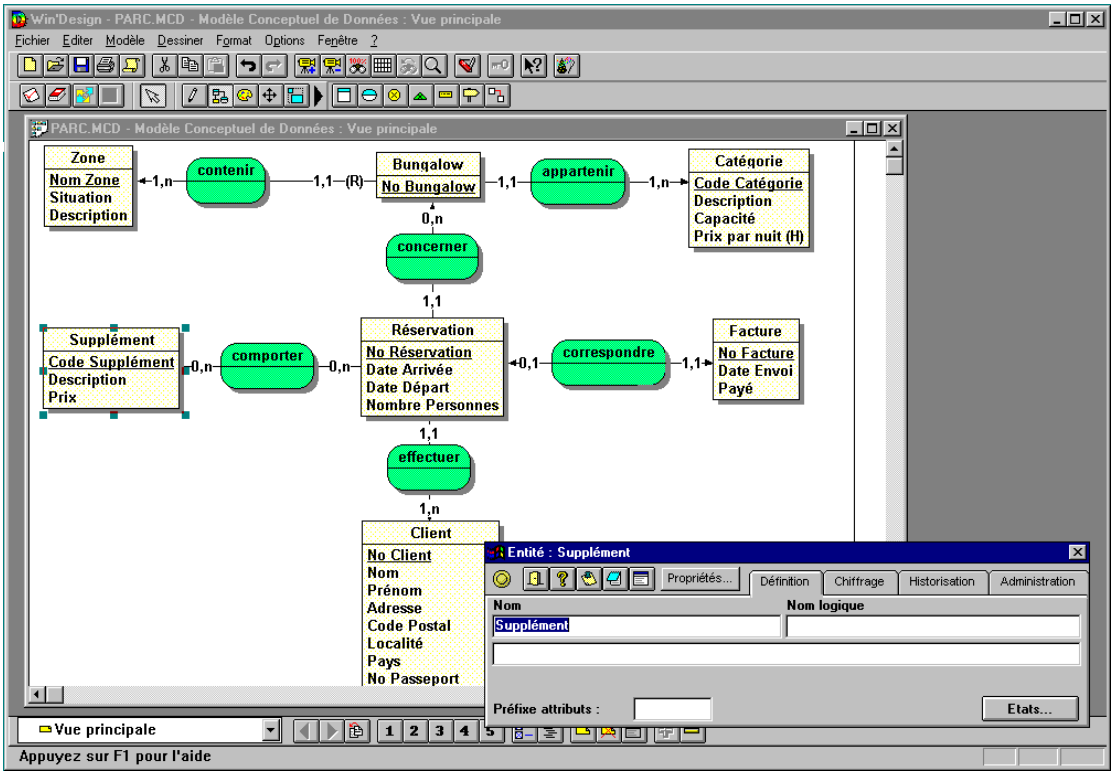
**Un outil de modélisation est un programme spécialisé dans le support de la conception d'un système d'information.**

Il existe actuellement sur le marché une offre très diverse d'outils de modélisation. Chaque outil de modélisation implémente une méthode de modélisation. Comme la méthode MERISE est très répandue dans nos régions, il est évident qu'il existe un certain nombre d'outils basés sur MERISE.

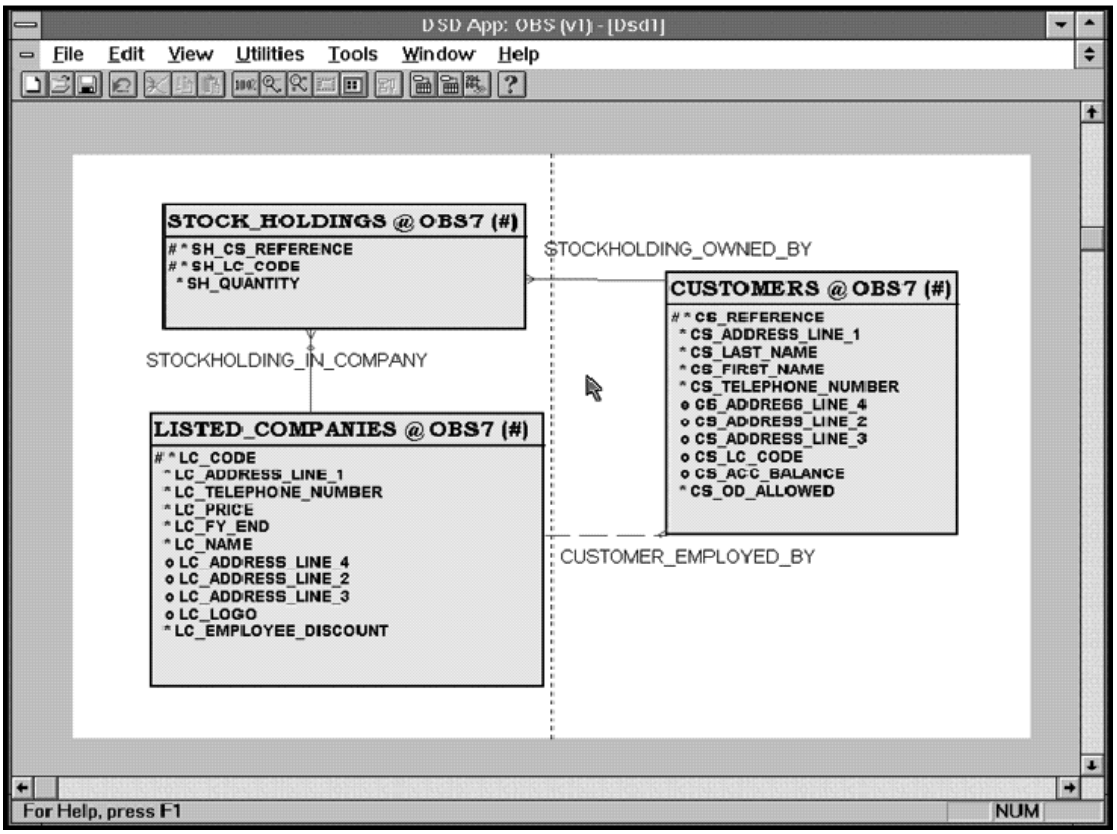
En principe, les outils de modélisation sont intégrés dans des applications capables de ne supporter pas uniquement la conception d'un système d'information (BD), mais également le développement complet de programmes de gestion d'une certaine envergure. Ces applications, appelées "**Ateliers de génie logiciel**" (angl. CASE Tool : Computer Aided Software Engineering Tool), sont généralement utilisés par les informaticiens afin de réaliser des grands projets.

Exemples:

L'outil **Win'Design** constitue une mise en œuvre de la méthode MERISE. Notons que Win'Design a été utilisé pour créer les modèles conceptuels et logiques présentés dans cet ouvrage.



L'application **Designer** de la société **Oracle,** constituant un atelier de génie logiciel très répandu, implémente une méthode plus ou moins compatible à MERISE en ce qui concerne la modélisation des données (Entity Relationship Modelling).



## 4.2 Fonctionnalités

Bien que les différents outils de modélisation, actuellement disponibles sur le marché, varient considérablement en termes de caractéristiques et fonctionnalités, ils offrent cependant les fonctions de base suivantes.

- Représentation graphique des modèles conceptuels et logiques avec les différents objets (p.ex. entités, relations, propriétés, identifiants, tables, attributs, clés etc.).
- Vérification des règles de construction des différents modèles (p.ex. Une relation ne peut pas être liée à deux entités via des cardinalités 1,1).
- Transformation automatique d'un MCD en MLD en respectant toutes les règles de transformation.
- Génération automatique d'une BD à partir d'un MLD. Après avoir indiqué le SGBD cible (p.ex. Oracle, MS-Access, Informix), le concepteur peut demander à l'outil de créer la BD. Pour ce faire, il existe deux alternatives:
  - l'outil de modélisation accède directement au SGBD cible afin de créer la BD en question;
  - l'outil de modélisation génère un script[1], qui est à la suite exécuté sur le SGBD afin de créer la BD.
- Génération automatique de rapports imprimés concernant l'état actuel d'un travail de conception. Ces rapports contiennent en général la représentation graphique des modèles, des listes avec tous les objets des différents modèles et des explications supplémentaires concernant certains objets.
- Gestion des objets de conception (p.ex. entités, relations, propriétés, identifiants, tables, attributs, clés etc.) dans un dictionnaire[2]. Pour des petits projets de conception, effectués par un seul concepteur sur un ordinateur, le dictionnaire est simplement un fichier stocké localement. Toutefois, pour les grands projets, effectués par plusieurs concepteurs, certains outils de modélisation permettent la gestion d'un dictionnaire sur un serveur en réseau (voir chapitre 5.5). Dans ce cas, plusieurs concepteurs peuvent travailler en même temps sur un modèle, l'outil de modélisation veillant à chaque moment que le modèle reste cohérent. L'intégration de plusieurs modèles en un seul modèle, et la gestion des versions d'un objet ou d'un modèle constituent d'autres caractéristiques supportées par un tel système.
- La plupart des outils de modélisation sont capables de créer un MLD et un MCD à partir d'une BD existante. Ce procédé, connu sous le nom de "Reversement d'une BD" (angl. Database Reverse Engineering), est souvent utilisé à la base d'un projet d'amélioration ou d'extension d'un système d'information existant déjà sous forme informatique.

[1] plusieurs commandes dans un langage supporté par le SGBD cible.
[2] une sorte de récipient logique pour les objets de conception.

# Partie 2 : Exploitation des bases de données relationnelles

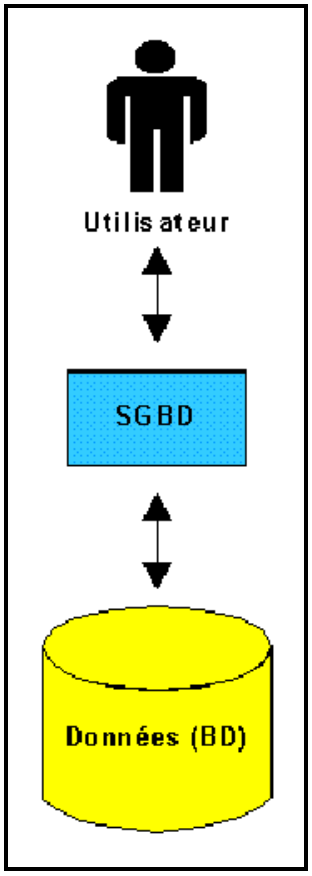
# 5. Les systèmes de gestion de bases de données

## 5.1 Définitions

**Une base de données (BD) est un ensemble bien structuré de données relatives à un sujet global. Ces données peuvent être de nature et d'origine différentes.**

Exemple: Une banque peut avoir une BD, qui contient les informations nécessaires sur tous les clients et leurs dépôts d'épargne.
Une société d'assurances peut stocker les données relatives aux contrats d'assurances ainsi qu'aux sinistres dans une BD.

**Un système de gestion de bases de données (SGBD) est un programme qui nous permet de créer, de modifier et d'exploiter des bases de données. Ce système constitue donc notre interface pour accéder aux données.**




**Un utilisateur utilise un SGBD pour accéder aux données d'une base de données.**

Par analogie:

Un utilisateur utilise un tableur pour accéder aux données d'une feuille de calcul, respectivement un traitement de texte pour accéder le texte d'un document.

**Exercice**

Discutez les avantages et désavantages d'une gestion de données informatisées à l'aide d'un SGBD, et comparez cette gestion à la gestion non informatisée.

## 5.2 Un peu d'histoire

Avant les années '70, la plupart des systèmes qui permettaient de gérer des grands volumes de données d'une façon plus ou moins cohérente étaient basés sur de simples fichiers séquentiels. Ces systèmes de gestion de fichiers (SGF) s'avéraient particulièrement limités lorsqu'il s'agissait de gérer une grande masse de données comportant des liens entre elles.

Classiquement, cette masse de données était répartie dans différents fichiers. L'utilisation de ces données n'était possible que par le biais de programmes spécialisés, qui ont du être réalisés par des programmeurs ayant une connaissance technique approfondie de la structure des fichiers. Chaque nouvelle interrogation du SGF nécessitait donc l'intervention d'un programmeur.

En plus, les SGF n'ont pas assuré la cohérence des données. Le programmeur était seul responsable pour garantir l'intégrité des données. Prenons l'exemple d'un SGF qui était utilisé dans une banque pour la gestion des clients et de leurs dépôts. Rien n'empêchait un programmeur de créer dans le fichier des dépôts un nouveau dépôt pour un client qui n'existait pas du tout dans le fichier des clients etc. .

Ceci étant seulement quelques exemples des inconvénients des SGF, nous remarquons qu'il était difficile pour un utilisateur d'utiliser directement un tel système. Il fallait souvent l'intervention d'un programmeur, qui devait faire bien attention à préserver la structure des données dans un rapport cohérent, tout en satisfaisant les besoins d'informations de l'utilisateur.

Déjà vers la fin des années 60, les premiers systèmes qui étaient capables de cacher la représentation interne des données à l'utilisateur, apparaissaient sur le marché. Ces systèmes, qui offraient à l'utilisateur une certaine structure logique pour stocker les données, étaient déjà équipés de certains mécanismes de base pour assurer la cohérence des données via des règles qui pouvaient être définies par l'utilisateur. Le système vérifiait ces règles lors de chaque modification des données. Dans un système de gestion des dépôts d'une banque, une telle règle pouvait par exemple exprimer le lien explicite entre un dépôt client et une personne. Ces systèmes étaient essentiellement basés sur les deux modèles de données suivants:

- Modèle réseau développé initialement par la "Conference On Data Systems and Languages" (CODASYL) en 1961
- Modèle hiérarchique développé pour la plus grande partie par la société IBM pendant les années 1965 - 1970

C'était en 1970 qu'un nouveau modèle pour représenter les données, le modèle relationnel, fut proposé par E.F.CODD. Le but de ce modèle, était d'accroître l'indépendance vis-à-vis de l'implémentation interne des données. Du point de vue de l'utilisateur, les données sont stockées dans un ensemble de tableaux, appelées "tables relationnelles" ou simplement "tables". Le stockage ainsi que la manipulation des données se basent sur le concept mathématique de l'algèbre relationnelle et du calcul relationnel. Ces concepts proviennent de la théorie mathématique des ensembles, et on y retrouve des notions telles que "Union", "Intersection" ou "Produit cartésien".

Il a fallu attendre le milieu des années 70 pour voir apparaître les premiers systèmes qui étaient basés sur le modèle relationnel, les "Systèmes de Gestion de Bases de Données Relationnelles (SGBDR)".

En 1976 apparaît le modèle Entité-Association, proposé par P.CHEN, qui donnait aux concepteurs des bases de données relationnelles une méthode adéquate pour modéliser des données d'un domaine quelconque (banques, assurances, industrie …) par une structure bien cohérente de tables relationnelles. Le modèle Entité - Association devenait ainsi le modèle théorique de conception de données sur lequel se basaient beaucoup de bases de données relationnelles.

Pendant les années '80 et '90, beaucoup de SGBDR étaient commercialisés pour les différentes plates-formes informatiques (Mainframe, Serveurs UNIX, Serveurs VMS, PC...). Citons quelques exemples de SGBDR populaires qui tournent actuellement sur PC:

- Personal ORACLE
- MS-ACCESS
- Visual dBASE
- Visual FOXPRO
- Borland PARADOX
- Lotus APPROACH

Aujourd'hui, les bases de données relationnelles se réjouissent d'une grande popularité. Surtout le domaine de la gestion des données à l'intérieur des entreprise est entièrement dominé par ces systèmes.

**Pour la suite de ce cours, nous allons nous limiter à l'étude des bases de données relationnelles. Nous entendons donc par chaque référence à une base de données (BD), la notion de base de données relationnelle. Il est également sous-entendu que les deux notions SGBD et SGBDR dénotent un système de gestion de bases de données relationnelles dans le contexte de ce cours.**

# 5.3 Les composants d'une base de données relationnelle

Une base de données relationnelle contient en général quatre types de composants (d'objets). Nous allons brièvement introduire ces types d'objets sans aller trop dans les détails. A chacun de ces objets sera consacré un chapitre séparé.

1. Les données sont stockées à l'intérieur de **tables**. Une table peut être comparée à une liste, qui contient des enregistrements relatifs à un domaine bien défini.
   Exemple: Le service du personnel de l'entreprise SCHAFFGAER S.à r.l. entretient une BD avec en outre une table pour les données des employées. Cette table contient un enregistrement pour chaque employé, avec le nom, le prénom, l'adresse, la localité, la date de naissance, la date d'entrée en service, le salaire mensuel et le nom du département auquel l'employé est actuellement affecté.

Employés : Table

| Nom | Prénom | Adresse | Localité | Date_naiss | Date_entrée | Salaire mensuel | Département |
|---|---|---|---|---|---|---|---|
| Witz | Evelyne | 24, Rue Gröhl | Grevenmacher | 24.02.1966 | 01.03.1996 | 65.000 Luf | Marketing |
| Midd | Erny | 10, Cité Liddrech | Luxembourg | 04.09.1959 | 01.01.1990 | 60.000 Luf | Comptabilité |
| Schlau | Suzette | 9, Av. A.Einstein | Esch-s-Alzette | 30.07.1971 | 15.10.1995 | 54.000 Luf | Marketing |
| Kuhl | Menn | 11, Cité A.Milk | Hupperdange | 27.02.1966 | 01.01.1990 | 78.000 Luf | Informatique |
| Super | Jhemp | 10, Rue Poznenö | Luxembourg | 23.08.1976 | 01.01.1996 | 69.500 Luf | Comptabilité |
| Boesch | Emil | 23, Am Hinterwald | Mamer | 23.08.1959 | 15.10.1996 | 49.000 Luf | Informatique |
| Tor | Vic | 1, Op der Areler Knippchen | Arlon | 22.07.1970 | 01.12.1996 | 70.000 Luf | Comptabilité |
| Vogel | Mätti | 2, Cité Perroquet | Luxembourg | 22.05.1970 | 15.08.1992 | 80.000 Luf | Service du personnel |
| Capon | Al | 3, Rue de la Gare | Luxembourg | 17.06.1969 | 23.08.1972 | 49.000 Luf | Comptabilité |
| Michel | Lyne | 1, Cité Gudjär | Colmar-Berg | 23.09.1966 | 01.01.1990 | 56.500 Luf | Service du personnel |

Record: 3 of 10

2. Les **requêtes** constituent dans un certain sens des "questions" qu'on pose au SGBD. Le résultat d'une requête est toujours un sous-ensemble d'une ou de plusieurs tables.
   Exemple: Le chef du personnel de l'entreprise SCHAFFGAER S. à r.l. désire connaître les noms, prénoms, adresses et localités des employés recrutés en 1996. Il doit formuler une requête qui sera exécutée par le SGBD, et qui donnera comme résultat une liste semblable à la table des employés, mais contenant uniquement les employés qui vérifient le critère de sélection de la requête, et pour chacun de ces employés seulement les informations demandées.

   Voici le résultat de cette requête:

Employés embauchés en 1996 : Select Query

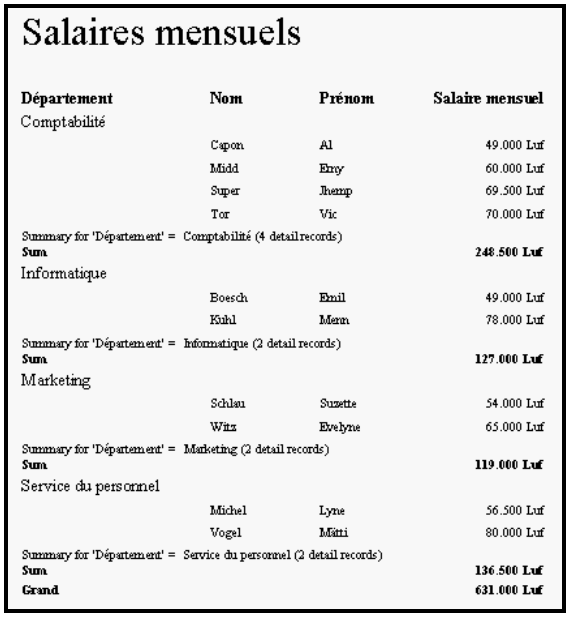
| Nom | Prénom | Adresse | Localité |
|---|---|---|---|
| Witz | Evelyne | 24, Rue Gröhl | Grevenmacher |
| Super | Jhemp | 10, Rue Poznenö | Luxembourg |
| Boesch | Emil | 23, Am Hinterwald | Mamer |
| Tor | Vic | 1, Op der Areler Knippchen | Arlon |

Record: 5 of 5

3. Les **formulaires** sont utilisés pour ajouter, modifier ou supprimer des données dans les tables. Bien que la plupart des SGBD nous permettent d'accéder les données directement dans les tables, les formulaires nous offrent certains avantages en ce qui concerne la facilité d'utilisation, mais également la sécurité des données.
   Exemple: La secrétaire du chef du personnel utilise un formulaire pour ajouter ou supprimer un employé de la BD. Ce formulaire lui permet également de modifier les données d'un employé.


4. Souvent on veut imprimer des statistiques; concernant certaines données d'une BD. C'est ici qu'interviennent les **rapports**. Les rapports sont similaires aux formulaires, à la différence près, qu'ils sont uniquement destinés à être imprimés et qu'il n'y a pas de dialogue interactif avec l'utilisateur. Un rapport se base généralement sur une ou plusieurs tables ou bien le résultat d'une requête.

   Exemple: A la fin de chaque mois le chef du personnel reçoit un rapport avec pour chaque département, la liste des employés, leur salaire mensuel ainsi que le salaire mensuel total payé par département.

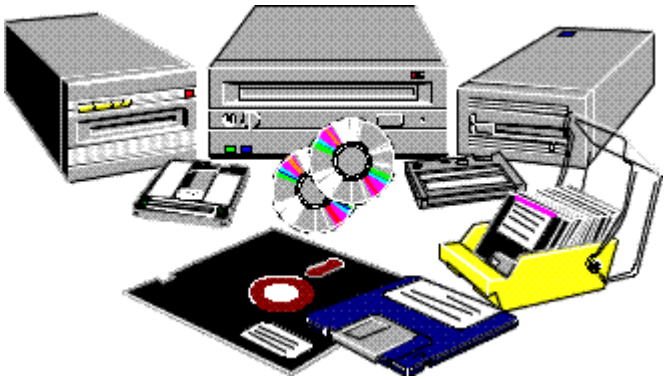


Salaires mensuels

| Département | Nom | Prénom | Salaire mensuel |
|---|---|---|---|
| Comptabilité | | | |
| | Capon | Al | 49.000 Luf |
| | Midd | Emy | 60.000 Luf |
| | Super | Jhemp | 69.500 Luf |
| | Tor | Vic | 70.000 Luf |
| Summary for 'Département' = Comptabilité (4 detail records) | | | |
| **Sum** | | | **248.500 Luf** |
| Informatique | | | |
| | Boesch | Emil | 49.000 Luf |
| | Kuhl | Mem | 78.000 Luf |
| Summary for 'Département' = Informatique (2 detail records) | | | |
| **Sum** | | | **127.000 Luf** |
| Marketing | | | |
| | Schlau | Suzette | 54.000 Luf |
| | Witz | Evelyne | 65.000 Luf |
| Summary for 'Département' = Marketing (2 detail records) | | | |
| **Sum** | | | **119.000 Luf** |
| Service du personnel | | | |
| | Michel | Lyne | 56.500 Luf |
| | Vogel | Mätti | 80.000 Luf |
| Summary for 'Département' = Service du personnel (2 detail records) | | | |
| **Sum** | | | **136.500 Luf** |
| **Grand** | | | **631.000 Luf** |

# 5.4 Structures physiques et logiques

Un SGBD utilise les ressources de l'ordinateur sur lequel il est exécuté. Les données des SGBD sont ainsi stockées par exemple sur un disque dur ou sur un autre support de stockage, de la même façon que les autres fichiers, générés par les autres programmes (p.ex. Traitement de texte, Tableur …) qui tournent sur l'ordinateur.



Qu'est-ce qu'on entend par structure physique ?

Le système d'exploitation (p.ex. DOS, Windows, Windows NT …), qui connaît seulement la notion de fichier en ce qui concerne le stockage des données, ignore en principe le contenu de ces fichiers. Les fichiers constituent dans un certain sens la structure physique des données.

Chaque programme crée des fichiers ayant un format spécifique à ce programme. L'utilisateur peut reconnaître le format par l'extension derrière le nom du fichier.

| | | | |
|---|---|---|---|
| Par exemple: | .doc | → | fichier MS-Word |
| | .xls | → | fichier MS-Excel |

Qu'est-ce qu'on entend par structure logique ?

Chaque programme offre également à l'utilisateur la possibilité de manipuler des composants, qui existent seulement dans le contexte de ce programme.

| | | | |
|---|---|---|---|
| Par exemple: | Document | → | Composant logique de MS-Word |
| | Feuille de calcul | → | Composant logique de MS-Excel |
| | Classeur | → | Composant logique de MS-Excel |

Ces composants ou structures logiques sont uniquement visibles par le biais du programme correspondant. On vient de définir les composants standard d'un SGBD dans le chapitre précédent:

- Les tables
- Les requêtes
- Les formulaires
- Les rapports

Quelle est la relation entre une structure logique et sa structure physique correspondante ?

Cette relation dépend du programme.

| | |
|---|---|
| MS-Word: | Chaque document (composant logique) correspond en principe à un fichier .doc (structure physique). |
| MS-Excel: | Chaque classeur (composant logique) correspond à un fichier .xls (composant physique). Attention: Un classeur peut contenir plusieurs feuilles de calcul. |

En ce qui concerne les SGBD, il existe deux variantes:

1. Chaque composant (table, formulaire …) d'une BD est stocké dans un fichier séparé. Une base de données constitue donc un ensemble de fichiers. Exemple: dBASE
2. Tous les composants d'une BD sont intégrés dans un seul fichier. Exemple: MS-Access

**Exercice**

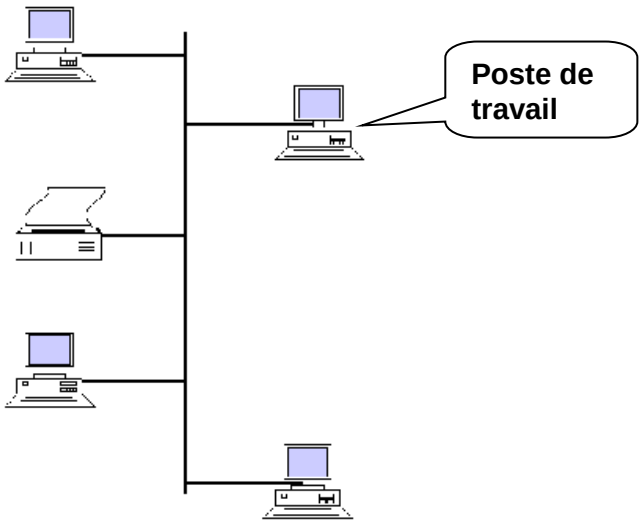
Discutez les avantages et désavantages des deux concepts d'implémentation possibles pour les composants d'une BD.

## A faire: TP No 1

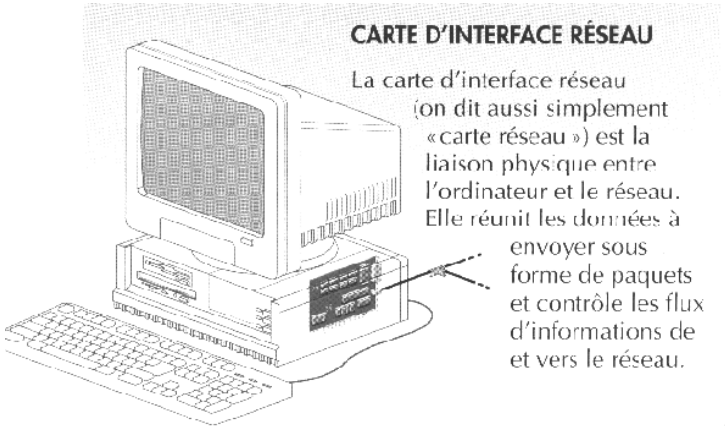
# 5.5 Les réseaux informatiques

**A son niveau le plus élémentaire, un réseau se compose de plusieurs ordinateurs reliés entre eux par un câble, afin de pouvoir échanger des données et partager des ressources, tels que des imprimantes, de l'espace disque etc. .**

Dans le contexte d'un réseau, ces ordinateurs sont appelés **postes de travail** (angl. workstation). Les postes de travail peuvent être répartis sur plusieurs étages d'un bâtiment ou même sur plusieurs bâtiments voisins. Un tel réseau est appelé **réseau local** (angl. LAN = Local Area Network).

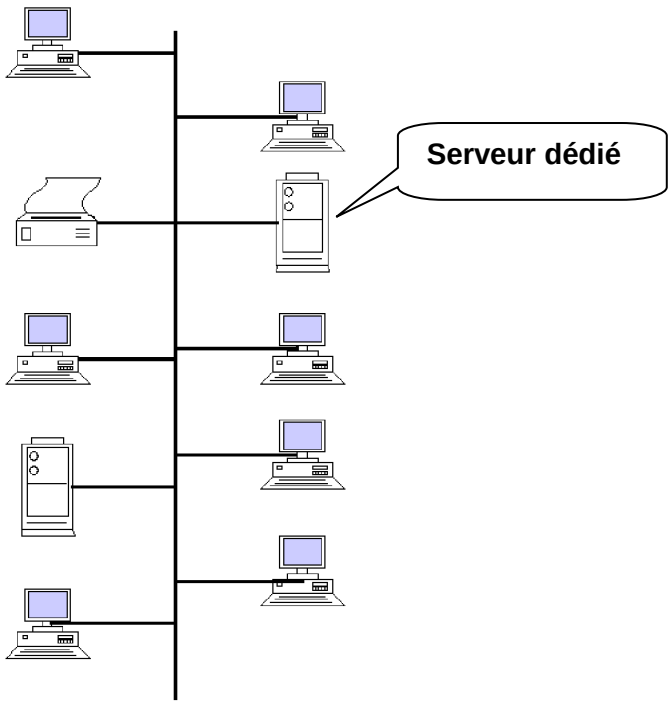


Afin de pouvoir être connecté à un réseau, un ordinateur doit disposer d'une **carte réseau**.




[1]

**La plupart des réseaux locaux contiennent des ordinateurs très puissants en termes de vitesse d'exécution et de capacité de stockage. Ces ordinateurs, encore appelés serveurs dédiés (angl. Server), ne sont généralement pas utilisés comme poste de travail, mais ils doivent effectuer un certain nombre de tâches variées.**




On distingue plusieurs types de serveurs.

- Les **serveurs de fichiers (angl. File Server)** contiennent généralement des fichiers appartenant aux différents utilisateurs du réseau. Par exemple, si vous utilisez un programme de traitement de texte sur un poste de travail, ce programme se trouve localement sur le poste. Cependant, le document sur lequel vous désirez effectuer des

[1] Extrait du livre "La Micro c'est simple", publié par IDG Books Worldwide, Inc. ISBN 2-87691-321-6

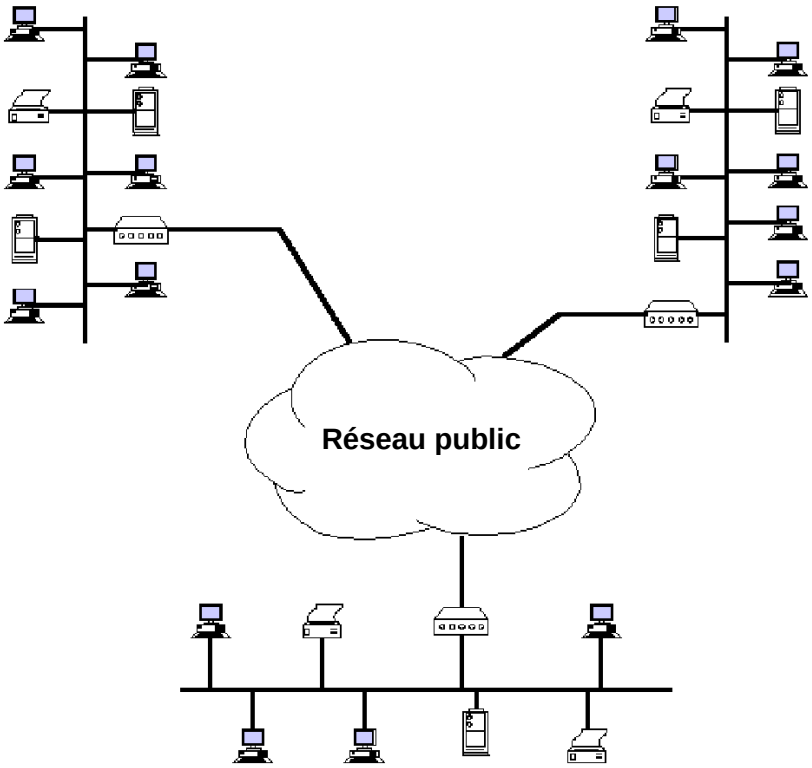
modifications, stocké sur le serveur, est chargé dans la mémoire locale de votre poste de travail, afin que vous puissiez l'utiliser. Lors de chaque opération de sauvegarde (angl. Save/Save As), le fichier est effectivement sauvegardé sur le serveur. Le serveur gère bien sûr l'accès des utilisateurs, qui doivent généralement s'identifier par un nom et un mot de passe, afin de garantir une certaine sécurité des données.

- Les **serveurs d'impression (angl. Print Server)** effectuent la gestion des imprimantes connectées au réseau. Lorsque le réseau comporte une multitude d'imprimantes différentes, un utilisateur sur son poste de travail peut sélectionner une imprimante en fonction des caractéristiques (p.ex. impression couleur/NB), des capacités (p.ex. nombre de pages imprimées par minute) et de l'emplacement physique (p.ex. imprimante à la même étage que le poste de travail). Lors de l'impression (angl. Print), le document à imprimer est d'abord envoyé dans une file d'attente (angl. Print Queue) qui se trouve sur le serveur d'impression. Le serveur d'impression contient généralement une file d'attente par imprimante. Les documents d'une file d'attente sont envoyés un après l'autre vers l'imprimante correspondante.

- Les **serveurs d'applications (angl. Application Server)** contiennent des applications ou programmes destinés à l'utilisation en réseau. Un exemple populaire constituent les applications du type "Groupware", qui permettent aux utilisateurs du réseau d'échanger des messages électroniques (angl. E-Mail), d'entretenir un agenda électronique commun et de travailler soi-disant en même temps sur des document partagés. Les **serveurs de bases de données (angl. Database Server),** appartenant également à cette catégorie, sont très répandues. Avant la période où les PC devenaient populaires, les bases de données ainsi que les programmes pour les manipuler, se trouvaient sur des grands ordinateurs puissants du type "Mainframe". L'utilisateur était connecté au mainframe à l'aide d'un terminal composé d'un clavier et d'un écran. Contrairement à un PC, un terminal peut uniquement envoyer des caractères au mainframe et afficher les caractères, qui lui sont envoyés par le mainframe. Avec l'arrivée des PC, dont les fonctionnalités ne se limitent pas à l'envoi et à l'affichage de caractères, le rôle des serveurs a considérablement changé. Actuellement, dans les environnements dits "Client/Serveur", les PC constituent des clients "intelligents", qui sont en parfaite communication avec les serveurs, dont le but principal est de "répondre" aux questions qui leurs sont posées par les clients. L'architecture Client/Serveur est explicitée plus en détail dans le chapitre 5.6 .

Les réseaux informatiques ayant une certaine taille, en termes du nombre de postes et de serveurs, sont généralement gérés par un **administrateur réseau**, personne (ou groupe de personnes) en charge de la gestion, du contrôle et de l'entretien du réseau.

Lorsque la distance géographique couverte par un réseau augmente en connectant des utilisateurs situés par exemple dans des villes ou même des pays différents, plusieurs réseaux locaux sont connectés en un seul **réseau étendu** (angl. WAN = Wide Area Network), qui peut ainsi regrouper plusieurs milliers d'utilisateurs.




En utilisant de l'équipement réseau spécialisé dans ce domaine, on peut connecter plusieurs réseaux locaux via un réseau public. Ce réseau public peut par exemple être constitué du réseau téléphonique public, d'un ensemble de lignes louées (lignes dédiées), d'un réseau de câbles en fibre optique, d'un réseau rapide de commutation de paquets ou même d'une liaison par satellite.

A titre d'exemple, on peut mentionner **l'Internet**, qui n'est rien d'autre qu'un gigantesque réseau étendu.

Pour un utilisateur, le **travail** dans un réseau local ou étendu est tout à fait **transparent**. Il peut par exemple accéder à des fichiers distants de la même manière qu'à des fichiers qui se trouvent sur son disque dur local.

# 5.6 L'approche Client/Serveur

## 5.6.1 La période des ordinateurs du type "Mainframe"

Avant la période où les PC devenaient populaires, les bases de données ainsi que les programmes pour les manipuler; se trouvaient sur de grands ordinateurs puissants du type "mainframe". On parlait d'une architecture centralisée, puisque les BD, le SGBD et les objets tels que requêtes, formulaires, rapports étaient stockés sur le "mainframe".

L'utilisateur était connecté au "mainframe" à l'aide d'un terminal composé d'un clavier et d'un écran. Contrairement à un PC, un terminal ne possède aucune "intelligence" propre, c.à.d. qu'il peut uniquement envoyer des caractères au "mainframe" et afficher les caractères, qui lui sont envoyés par le "mainframe", et ceci en plus uniquement en mode caractère[1].

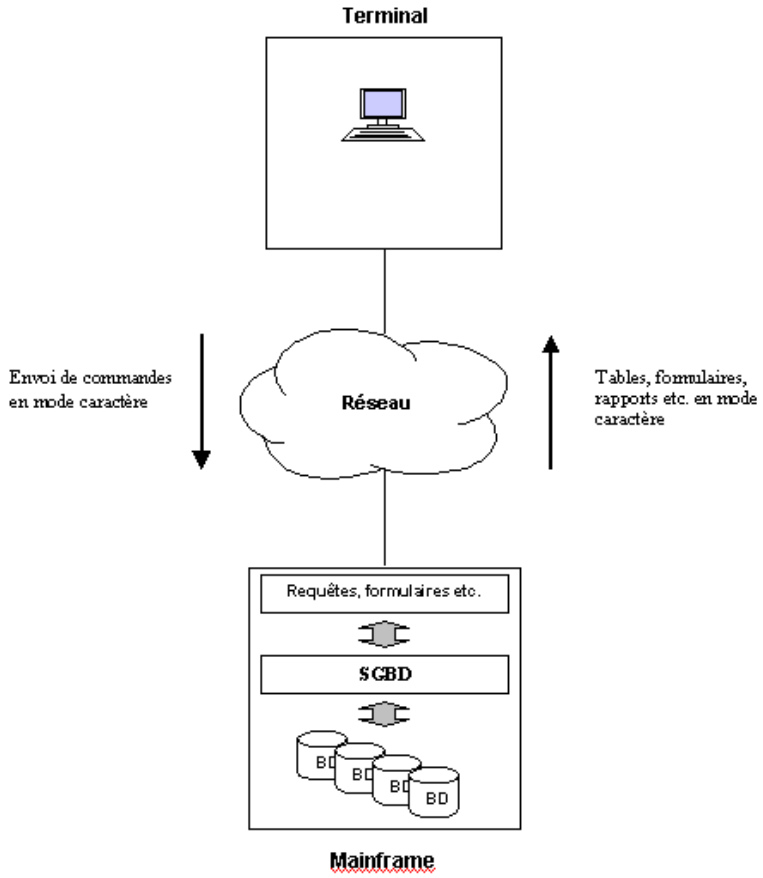
Lorsque l'utilisateur veut par exemple afficher un formulaire, la construction du formulaire se fait complètement sur le "mainframe". Ensuite, le formulaire où plutôt l'apparence du formulaire est envoyé via le réseau vers le terminal de l'utilisateur.

Exemple d'un formulaire en mode caractère:

```
CLSCH73   08:44:49   *** COMPTANT CLIENT ***       NUMERO POLICE:  PROJET    F
EFFET: 00 00 2000      EXPIRATION: 16 03 1995      ECHEANCE: 16 03 COR:
PERIODE DE DECOMPTE A LA BASE DE CE MOUVEMENT: DU 16 03 1994 AU 16 03 1995
DERNIERE EMISSION: 00 00 2000       JOURS:  PROD.   360  ANNUL.     0
S.-BRANCHE PRORATA  ANC.    NOUV.  COMPTANT   BONUS    IMPOTS       TOTAL
INCENDIE              0    11395    11395         0       458       11853
TEMPETE               0     3798     3798         0       152        3950
VOL                   0      682      682         0        27         709
D.-EAUX               0     6489     6489         0       260        6749
R.-CIVILE             0      928      928         0        37         965
ATTENTATS             0      678      678         0        27         705
DEF.-REC.             0       90       90         0         4          94
ASSISTANCE            0      226      226         0         9         235
T O T A L             0    24286    24286         0       974       25260
```

[1] un terminal est incapable d'afficher des graphiques

Architecture "mainframe":




Avantages de l'architecture "mainframe":

- Les "mainframe" étant de grands ordinateurs très puissants, les systèmes atteignent de très belles performances, d'autant plus qu'il n'y a pas de représentation graphique sur les terminaux.

Désavantages de l'architecture "mainframe":

- Aucune capacité de calcul sur le terminal, donc impossible d'exécuter des programmes sur le terminal.
- Pas de représentation graphique sur le terminal. Formulaires etc. moins faciles à utiliser.
- Le "mainframe" étant sous la seule gestion du service informatique, les utilisateurs peuvent uniquement accéder les BD via des formulaires etc. créés par les informaticiens. (Cette mesure s'avère parfois avantageuse ☺)
- Le réseau est assez chargé, surtout lorsque le nombre de terminaux accroît.
- Les requêtes, formulaires etc. sont fortement couplés au SGBD ce qui les rend pratiquement inutilisable lorsqu'une société veut migrer vers un autre SGBD.

Exemples de SGBD pour "Mainframe":

- DB2 de IBM
- RDB de DEC

En fait, les informaticiens sont depuis longtemps à la recherche de systèmes ouverts. La finalité d'un système ouvert consiste dans le fait que ses composants (ordinateurs, SGBD etc.) sont échangeables sans que tous les objets en utilisation (requêtes, formulaires etc.) doivent être complètement redéfinis resp. reprogrammés. Tous les éléments d'un tel système doivent donc supporter au maximum possible certains standards. Un élément important de la philosophie des systèmes ouverts est constitué par l'approche Client/Serveur.

## 5.6.2 L'approche Client/Serveur

L'évolution historique des architectures informatiques vers les architectures du type Client/Serveur (angl. Client/Server) dans les années '90; peut être ramenée surtout aux facteurs suivants.

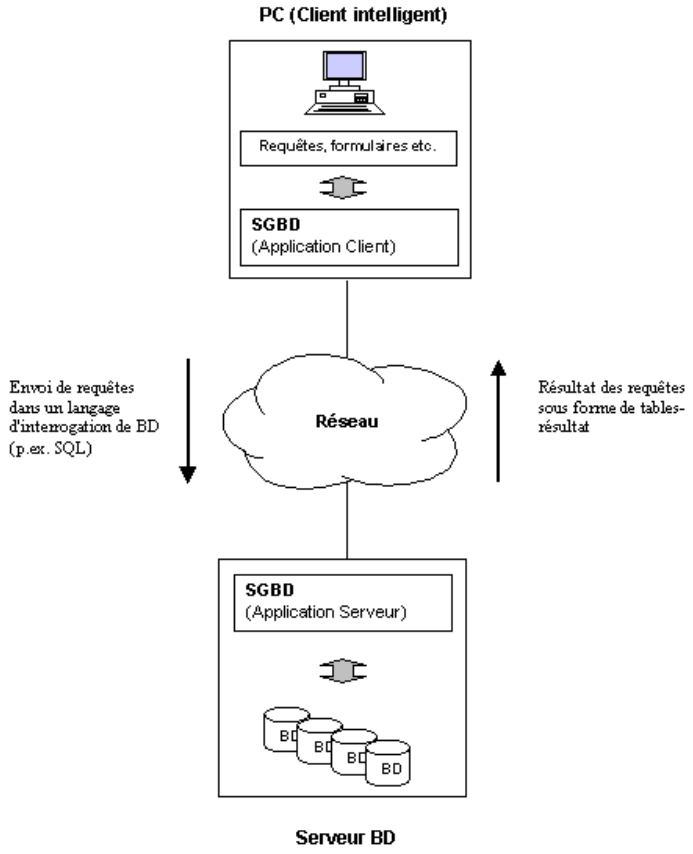
- L'arrivée au marché des PC.
- L'apparition de serveurs, machines moins chères et moins spacieuses que les "mainframe", avec cependant une capacité de calcul et de stockage analogue à celle des "mainframe".
- L'émergence de systèmes d'exploitations standardisés tels que UNIX ou Windows NT.
- L'apparition des SGBD indépendants de la plate-forme[1] et disponible pour tous les systèmes d'exploitation standardisés

L'approche Client/Serveur implémente une décentralisation des applications BD. En fait, les BD sont gérées sur un serveur BD, tandis que les interfaces pour visualiser et manipuler les données (p.ex. formulaires, rapports) se trouvent sur les PC client, dans un environnement ergonomique[2].

Sur le poste client se trouve donc en principe un SGBD client, offrant toutes les fonctionnalités requises, qui émet des requêtes formulées dans un langage d'interrogation de données[3] au serveur BD via le réseau. Le serveur exécute les requêtes qui lui ont été transmises et renvoie le résultat au client. Le client représente alors le résultat en se servant par exemple d'un formulaire ou d'un rapport qui a été défini antérieurement.

[1] par plate-forme, on entend l'ordinateur sur lequel est exécuté le SGBD
[2] plus facile à utiliser
[3] par exemple SQL (voir chapitre 7.2)

Architecture Client/Serveur:




Avantages de l'architecture Client/Serveur:

- Les utilisateurs deviennent des clients avec des postes de travail intelligents (PC), à l'aide desquels ils peuvent connecter les applications bureautiques directement aux serveurs BD, afin de gérer dans un environnement convivial les données de l'entreprise, sans être dépendant des services d'un informaticien pour résoudre le moindre problème.
- Les réseaux informatiques modernes permettent un accès transparent à plusieurs serveurs BD, et ceci même de façon simultanée.
- Une partie de la capacité de travail est partagée entre les serveurs et les clients, ce qui crée un certain équilibre.
- Une panne du serveur n'empêche pas nécessairement tous les utilisateurs de travailler avec l'outil informatique. Certains travaux peuvent être exécutés sans connexion au serveur.
- L'architecture Client/Serveur, reposant sur les systèmes ouvert, offre en plus l'avantage qu'il existe toute une panoplie de logiciels standards, ce qui crée un marché multivendeur et une offre de produits équilibrée.

Exemples de SGBD Client/Serveur:

- Côté serveur: Oracle, Sybase, Informix, MS-SQL-Server
- Côté client: BORLAND Paradox, Personal Oracle, MS-Access

# 6. Les tables (angl. tables)

## 6.1 Définition

⚠ Une table est une **collection de données** relatives à un domaine bien défini, par exemple les employés d'une société ou les livres d'une bibliothèque. Elle contient des enregistrements dont chacun est composé par les mêmes champs de données.

Voici, à titre d'exemple, quelques employés d'une société:

Jos Weber

Age: 34

Salaire: 68000 Luf.

Service: Comptabilité

Antonio Da Costa

Age: 27

Salaire: 70000 Luf.

Service: Informatique

Emil Feller

Age: 43

Salaire: 65000 Luf.

Service: Expédition

. . .

Voici la table nécessaire pour stocker les informations concernant ces employés dans une BD:

**Table : tblEmployés**

**Un champ de données** ↓

| Nom | Prénom | Age | Salaire | Service |
|---|---|---|---|---|
| Weber | Jos | 34 | 68000 Luf | Comptabilité |
| Da Costa | Antonio | 27 | 70000 Luf | Informatique |
| Feller | Emil | 43 | 65000 Luf | Expédition |
| ..... | ..... | ..... | ..... | ..... |

← **Un enregistrement**

⚠ **Propriétés des tables:**

- **Les champs de données définissent les informations, qu'on veut stocker dans la table** (p.ex. des informations concernant les employés d'une société)**.**
- **Chaque enregistrement représente une occurrence de ce qu'on veut stocker** (→ p.ex. un employé)**.**
- **Chaque table possède un nom unique** (p.ex. : *tblEmployés*)**.**
- **Chaque enregistrement correspond à une ligne de la table.**
- **Chaque champ correspond à une colonne de la table.**
- **Chaque champ peut représenter des données de nature différente** (Nom, Salaire, Date de naissance …)**.**
- **Chaque champ peut représenter des données de type différent** (Texte, Nombres, Dates …)**.**

**Convention des noms:**

Il existe une convention concernant les noms des objets des BD. Généralement, les noms des objets ne contiennent ni d'espaces, ni de caractères spéciaux. En plus, chaque nom d'un objet est précédé par un préfixe bien déterminé pour chaque type d'objet. Cette convention fait partie d'une convention des noms générale pour les programmes tournant sous une interface graphique du type Windows.

Les noms de tables sont précédés du préfixe **tbl** (angl.: table).
Par exemple: tblLivres, tblEmployés

**Le nom d'une table doit être unique à l'intérieur d'une BD.**

Une BD peut contenir une ou plusieurs tables, mais les tables sont généralement la condition nécessaire pour la création d'autres objets tels que les requêtes, formulaires et rapports.

**Exercice**

Déterminez les champs nécessaires pour une table qui contiendra des données concernant :
- les élèves d'une école (nous ignorons la gestion des classes);
- les livres d'une bibliothèque (nous supposons qu'un livre est rédigé par un seul auteur);
- les produits d'un supermarché.

# 6.2 Les champs d'une table

Une table reprenant les données concernant les voitures d'une société de taxis contient par exemple pour chaque enregistrement (= chaque taxi) les informations suivantes:

- Marque
- Modèle
- Cylindrée
- Poids


Il est évident que les informations sont de types différents. Tandis que la marque et le modèle sont représentés par des chaînes de caractères (p.ex. "Ford", "BMW", …), la cylindrée et le poids sont représentés par des valeurs numériques.

Voici, à titre d'exemple, une table qui représente les taxis dans une BD:

| **Marque** | **Modèle** | **Cylindrée** | **Poids** |
|---|---|---|---|
| BMW | 525i | 2500 | 1360 |
| Ford | Orion | 1800 | 1080 |
| BMW | 320i | 2000 | 1200 |
| …. | … | … | … |

Afin de pouvoir représenter des données de types différentes, les SGBD offrent des types de données standards pour les champs de données. Voici les types de données connus par la plupart des SGBD:

| **Type de données** | **Description** |
|---|---|
| Date/Heure | Date et heure |
| Valeur booléenne | Seulement les 2 valeurs *Oui/Non* (Yes/No) sont possibles |
| Texte | Chaînes de caractères |
| Numérique | Nombres entiers ou décimaux |
| Mémo | Documents (textes long) |

Consultez le manuel d'utilisation de votre SGBD pour trouver des informations plus détaillées concernant les types de données supportés.

| | |
|---|---|
| Remarque: | Les nombres qui ne sont pas utilisés lors de calculs numériques (p.ex. No.Tél) sont généralement représentés à l'aide du type de données "Texte". |

**Convention des noms:**

Les noms des champs sont précédés du préfixe **fld** (angl.: field).
Par exemple: *fldMarque, fldModèle* …

**Exercice**

Réfléchissez pour chaque champ des 3 tables, que vous avez défini dans l'exercice du chapitre 6.1, sur le type de données approprié.



**Exercice**

Détaillez dans la table suivante, pour chaque type de données implémenté par votre SGBD, le nom ainsi qu'une description. Référez vous au manuel d'utilisation ou au système d'aide en ligne.

| Type de données | Description |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

**Lors de la création d'une table, nous devons indiquer au SGBD, pour chaque champ:**

1. **Le nom du champ, qui doit être unique dans la table**
2. **Le type de données du champ**

## 6.3 Clé primaire

Dans la plupart des cas, on désire pouvoir identifier de manière unique chaque enregistrement de la table. Ceci n'est pas possible pour notre table avec les taxis. Il se peut très bien que le propriétaire de la société achète par exemple une deuxième BMW 320i , qui possède bien sur également une cylindrée de 2000 ccm et un poids de 1200 kg. Dans ce cas nous avons 2 enregistrement complètement identiques dans notre BD. Cela nous empêche d'identifier clairement un des 2 enregistrements.

Il nous faut donc un moyen, qui nous permet d'adresser sans ambiguïté chaque enregistrement dans la table → une clé primaire !

**La clé primaire, constituée d'un ou de plusieurs champs, nous permet d'identifier de manière unique chaque enregistrement d'une table.**

Examinons notre cas de la société de taxis. Aucun des 4 champs seuls, et aucune combinaison des 4 champs ne se prêtent comme candidats pour devenir clé primaire, car aucun de ces champs ne contient des valeurs uniques à un et un seul taxi. Supposons par exemple la marque et le modèle comme clé primaire. Au cas ou la société achète un deuxième BMW 320i, on ne pourrait plus distinguer entre les deux voitures.

Le ou les champs, qui forment la clé primaire doivent impérativement avoir des valeurs qui sont uniques pour toute la table[1], et qui permettent donc d'identifier chaque enregistrement.

Exemples:
Le numéro de la matricule pour les assurés des caisses de maladie.
Le numéro client pour les clients d'une vidéothèque.

En ce qui concerne les taxis, nous avons deux possibilités:

1. Analyser s'il n'existe pas d'information concernant les taxis qui ne soit pas encore stockée dans la table et qui ferait une clé primaire valable. Une telle information serait par exemple le numéro de chassis, unique pour chaque voiture. On pourrait donc ajouter un champ *fldNochassis* et définir ce champ comme clé primaire. Ceci a comme désavantage que le numéro de chassis d'une voiture est un numéro assez long et compliqué, ce qui défavorise une utilisation conviviale de la table.

2. On pourrait inventer un numéro de taxi allant simplement de 1 jusqu'au nombre de taxis que la société possède. Le premier taxi enregistré serait le numéro TAXI=1, le deuxième le numéro TAXI=2 etc. . Bien que ce numéro n'ait aucune signification réelle, cette méthode de création de clés primaires artificielles est très répandue, et la plupart des SGBD offrent même un type de données prédéfini pour générer des valeurs uniques pour de telles clés primaires. Notre table aurait dans ce cas la structure suivante:

[1] pour une clé primaire composée de plusieurs champs, la combinaison des valeurs doit être unique

## Clé primaire

| idTaxi | fldMarque | fldModèle | fldCylindrée | fldPoids |
|---|---|---|---|---|
| 1 | BMW | 525i | 2500 | 1360 |
| 2 | Ford | Orion | 1800 | 1080 |
| 3 | BMW | 320i | 2000 | 1200 |
| ... | ... | ... | ... | ... |

**Convention des noms:**

Les noms des champs qui forment la clé primaire sont précédés du préfixe **id** (angl.: identifier).
Par exemple: idTaxi, idEmployé

**Exercice**

Définissez pour chacune des 3 tables, que vous avez défini dans l'exercice du chapitre 6.1, une clé primaire parmi les champs existants, resp. créez un nouveau champ qui assumera le rôle de clé primaire. Indiquez dans la grille suivante pour chaque table toutes les informations nécessaires.

| Nom de la table: | | | |
|---|---|---|---|
| Membre de la clé primaire (Cochez la case si OUI ) | Nom du champ | Type de données | Description |
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |

| **Nom de la table:** | | | |
|---|---|---|---|
| **Membre de la clé primaire (Cochez la case si OUI )** | **Nom du champ** | **Type de données** | **Description** |
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |

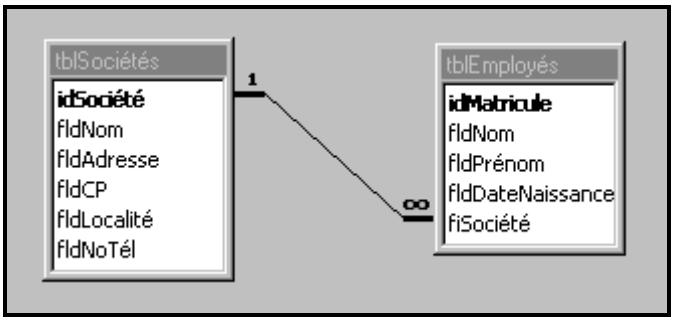
| **Nom de la table:** | | | |
|---|---|---|---|
| **Membre de la clé primaire (Cochez la case si OUI )** | **Nom du champ** | **Type de données** | **Description** |
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |
| ☐ | | | |

# 6.4 Relations entre tables - clé étrangère

Une base de données bien concipée est rarement composée d'une seule table, mais d'un ensemble de tables, entre lesquelles il existe certaines relations (voir Partie1: Modélisation d'un système d'information).

Exemple:

Soit la BD suivante d'un organisme de sécurité sociale.




La table *tblEmployés* contient certaines informations concernant les employés, mais pas le nom de la société, qui emploie un employé en question. Les informations des sociétés se trouvent dans la table *tblSociétés*. Cependant, dans la table *tblEmployés* se trouve le champ *fiSociété*, qui contient pour chaque employé le numéro de la société patron. On peut retrouver chaque numéro de société encore une fois dans le champ *idSociété*, qui constitue la clé primaire de *tblSociétés*.
Les deux tables sont donc **logiquement liées** via les champs *fiSociété* et *idSociété*.

On dit que *fiSociété* est une **clé étrangère,** qui fait référence à la **clé primaire** *idSociété* de la table *tblSociétés*.

**Clé étrangère**

**Un champ qui, dans une table, fait référence à la clé primaire d'une autre table est appelé clé étrangère (angl.: foreign key). Ainsi sont définies les relations entre les tables.**

# 6.5 Index

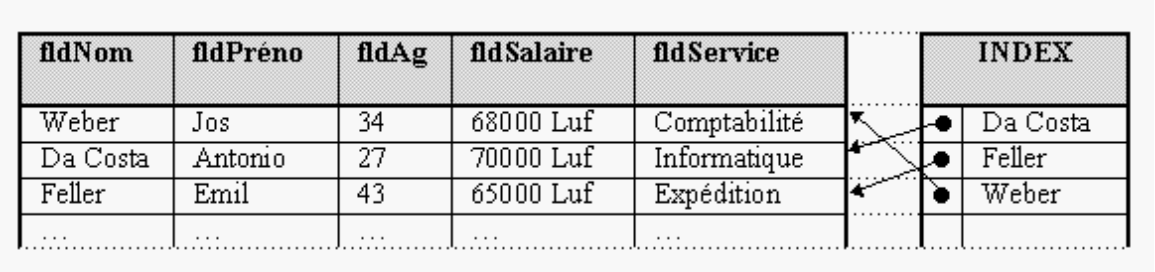
Une des utilisations fréquentes des tables consiste dans la recherche et le tri des enregistrements.

Lorsque les tables contiennent un grand nombre d'enregistrements, la recherche de certains enregistrements ainsi que le tri d'enregistrements nécessitent de plus en plus de temps. Les index sont des structures qui accélèrent les tris et recherches dans les tables, ainsi que l'exécution de certaines requêtes (voir chapitre 7).

Exemple:

Reprenons notre exemple des employés d'une société. Une recherche intéressante serait par exemple: MONTRE-MOI TOUS LES EMPLOYES DU SERVICE INFORMATIQUE !
Il serait aussi intéressant de trier les employés sur leur nom de famille. Au cas ou la table contient beaucoup d'enregistrements, on devrait d'abord créer un index sur le champ **fldNom**, afin d'accélérer le tri.

Créer par exemple un index sur le champ **fldNom** veut dire que le SGBD copie toutes les valeurs existantes du champ **fldNom** dans une liste spéciale à 2 colonnes. La deuxième colonne contient les noms triés en ordre alphabétique, et la première contient une référence vers l'enregistrement correspondant de la table.




Il est évident que par la suite de la création de cet index, toutes les recherches et les tris concernant le nom de l'employé sont accélérées, puisque le SGBD consulte uniquement l'index pour retrouver le bon nom, pour ensuite utiliser la référence de l'index vers l'enregistrement correspondant de la table.

Un index peut aussi comporter plusieurs champs comme par exemple **fldService** et **fldNom**.

**Propriétés importantes des index:**

- Un index est toujours lié à un ou plusieurs champs d'une table.
- Un index peut seulement contenir des champs ayant un des types de données Texte, Numérique ou Date/Heure.
- Un index est automatiquement mis à jour par le SGBD lors d'un ajout, d'une modification ou d'une suppression d'enregistrements dans la table. Ceci est transparent pour l'utilisateur de la BD.
- Il existe deux types d'index:

1. Index avec doublons (Les valeurs doubles sont permises)
2. Index sans doublons (Les valeurs doubles ne sont pas permises)

Voici quelques règles qui nous aident à déterminer les champs d'une table qui ont besoin d'être indexés:

- La puissance des index joue uniquement pour des tables qui contiennent beaucoup d'enregistrements (Consultez la documentation de votre SGBD afin d'avoir des précisions).
- Un champ sur lequel on ne fait que rarement ou pas du tout de recherche ou de tri n'a pas besoin d'index.
- Les champs référencés fréquemment dans les recherches et tris doivent par contre être indexés.
- Pour les index multi-champs, il faut veiller à ce que la combinaison des champs dans l'index corresponde exactement au critère de recherche. Un index sur **nom&prénom** n'accélère pas une recherche du type **prénom=Jos & nom=Weber.**
- Un index sans doublons sur un champ empêche l'utilisateur d'entrer la même valeur dans ce champ, dans deux enregistrements différents.
- Définir trop d'indexes sur une table ralentit en général les opérations d'ajout, de modification et de suppression, parce que le SGBD doit mettre à jour la table **et** l'index.

 **La clé primaire est toujours indexée à l'aide d'un index sans doublons ! [1]**

 **A faire : TP No 2**

[1] Pour la plupart des SGBD, ceci est fait de façon automatique lors de la définition d'un ou de plusieurs champs comme clé primaire .

# 7. Les requêtes (angl. queries)

## 7.1 Définition

Nous avons vu que la plupart des SGBD offrent la possibilité d'effectuer des recherches directement dans les tables. Les possibilités de formuler des critères de recherche sont cependant souvent assez limitées. Heureusement, la plupart des SGBD nous offrent également la possibilité de poser pratiquement n'importe quelle "question" à nos tables, sous forme de requêtes.

Les requêtes servent donc à répondre aux questions basées sur le contenu d'une ou de plusieurs tables. Nous allons plus tard étudier des requêtes, qui se basent sur plusieurs tables, mais pour l'instant nous allons nous limiter aux questions simples basées sur une seule table.

Exemple:

Reprenons notre table avec les taxis:

| **Taxi** | **fldMarque** | **fldModèle** | **fldCylindrée** | **fldPoids** |
|---|---|---|---|---|
| 1 | BMW | 525i | 2500 | 1360 |
| 2 | Ford | Orion | 1800 | 1080 |
| 3 | BMW | 320i | 2000 | 1200 |
| | …. | … | … | … |

Une requête simple serait par exemple:

- Quelles sont les marques et modèles des voitures ayant une cylindrée supérieure à 2000 ?

Le résultat serait un sous-ensemble de la table avec seulement les enregistrements qui vérifient le critère de sélection (→ cylindrée > 2000). Pour chacun de ces enregistrements, le SGBD affiche en plus seulement les champs explicitement demandés (→ *fldMarque* et *fldModèle*).

Une requête simple produit donc comme résultat un sous-ensemble des enregistrements d'une table. En plus, une requête nous permet d'afficher seulement certains champs pour les enregistrements appartenant à ce sous-ensemble.

On appelle ces requêtes "Requêtes de Sélection", puisqu'il s'agit d'une sélection de certains enregistrements.

Il n'existe cependant pas seulement des requêtes de sélection, mais également:

- Des requêtes d'insertion → insérer des enregistrements dans la table.
  p.ex. Insérer un nouveau taxi : 4, BMW, 325i, 2500, 1270.

- Des requêtes de modification → modifier des enregistrements dans la table.
  p.ex. Modifier la cylindrée des Ford Orion de façon à ce qu'elle devienne 2000.

- Des requêtes de suppression → supprimer des enregistrements de la table.
  p.ex. Supprimer toutes les BMW.

Les requêtes possèdent l'avantage de pouvoir manipuler facilement un grand nombre d'enregistrements sans que l'utilisateur ne doive s'occuper de sélectionner enregistrement par enregistrement. Il lui suffit de spécifier des critères de sélection pour la requête, ainsi que l'opération à effectuer (→ simple sélection et affichage, insertion, modification ou suppression).

Bien que les requêtes de sélection soient implémentées d'une manière plus ou moins cohérente à travers les SGBD actuels, il existe des différences subtiles en ce qui concerne les requêtes d'insertion, de modification ainsi que de suppression. En plus, l'insertion et la suppression se font souvent de manière plus facile directement dans la table.

**Il existe 4 types de requêtes:**

1. **Requêtes de sélection.**
2. **Requêtes d'insertion.**
3. **Requêtes de modification.**
4. **Requêtes de suppression.**

Pour chaque requête nous retrouvons le cycle suivant:


**Exercice**

Quel est en général le résultat d'une requête:

- de sélection :

- d'insertion :

- de modification :

- de suppression :

# 7.2 Introduction au langage SQL

## 7.2.1 Généralités

Nous avons vu au chapitre précédent qu'il faut d'abord formuler une requête et puis l'exécuter, afin d'avoir des résultats. Vous pouvez probablement bien vous imaginer que les SGBD actuels ne comprennent pas le langage naturel. Aucun SGBD n'offre une possibilité d'écrire p.ex. *Je veux voir tous les taxis dont la marque est Ford* Pour formuler une requête, l'utilisateur doit donc utiliser un langage spécialisé pour ce domaine.

Le langage **SQL (Structured Query Language)** est un **standard international**, en ce qui concerne les langages de manipulation des BD. SQL est connu par tous les SGBDR. Il faut cependant mentionner que, désormais la présence de standards internationaux tels que SQL-86, SQL-89 ou SQL-92, chaque SGBD sur le marché utilise un peu son propre dialecte du langage SQL.

## 7.2.2 Syntaxe SQL de base

Nous distinguons les 4 types de requêtes suivants.

**1. Requêtes de sélection.**

```
SELECT <Nom des champs>
FROM <Nom de la table>
WHERE <Critères de sélection>;
```

**2. Requêtes d'insertion.**

```
INSERT INTO <Nom de la table> [(<Liste des champs>)]
VALUES ( <Valeurs pour les champs> );
```

Attention: Lorsque vous n'indiquez pas la liste des champs derrière INSERT INTO , vous devez spécifier une valeur pour chaque champ de la table derrière VALUES . Les parenthèses derrière VALUES sont obligatoires. La liste des champs, lorsqu'elle est indiquée, contient les noms des champs, séparés par une virgule, et doit également être entourée de parenthèses.

**3. Requêtes de modification.**

```
UPDATE <Nom de la table>
SET <Nom d'un champ>={valeur}, <Nom d'un champ>={valeur}, . .
WHERE <Critères de sélection>;
```

**4. Requêtes de suppression.**

```
DELETE FROM <Nom de la table>
WHERE <Critères de sélection>;
```

Soit une table des employés d'une entreprise avec la structure suivante:

**tblEmployés**

| Nom du champ | Type de données | Description |
|---|---|---|
| idEmployé | Numérique | Numéro de l'employé (Clé primaire) |
| fldPrénom | Texte | Prénom de l'employé |
| fldNom | Texte | Nom de l'employé |
| fldNationalité | Texte | Nationalité de l'employé |
| fldAge | Numérique | Age de l'employé |
| fldSexe | Texte | Sexe de l'employé |
| fldService | Texte | Service auquel l'employé est affecté |

Voici le code SQL nécessaire pour effectuer quelques requêtes élémentaires:

1. Afficher le prénom et le nom de tous les employés

```
SELECT fldPrénom, fldNom
FROM tblEmployés;
```

2. Insérer une nouvelle employée:

| | | | | | | |
|---|---|---|---|---|---|---|
| 20 | Angela | Portante | ITA | 27 | F | Comptabilité |

```
INSERT INTO tblEmployés
VALUES (20, 'Angela', 'Portante', 'ITA', 27, 'F', 'Comptabilité');
```

**Remarques:**

**Les valeurs sont séparées par des virgules.**

**En SQL, les données du type TEXTE sont entourées d'apostrophes.**

**Les dates sont entourées du caractère # et indiquées dans le format américain #Mois/Jour/Année#**
**Exemple: 20/4/98 → #04/20/98# ou #4/20/98# ou #04/20/1998# ou #4/20/1998#**

3. Afficher toutes les nationalités représentées dans la société

```
SELECT fldNationalité
FROM tblEmployés;
```

Quel est inconvénient de cette requête ? _____

Soit l'adaptation suivante de la requête:

```
SELECT DISTINCT fldNationalité
FROM tblEmployés;
```

Expliquez l'utilité de l'option **DISTINCT**

4. Afficher tous les champs pour tous les employés

```
SELECT idEmployé, fldPrénom, fldNom, fldNationalité, fldAge, fldSexe, fldService
FROM tblEmployés;
```

ou

```
SELECT *
FROM tblEmployés;
```

**Remarque:**

**L'opérateur * permet d'afficher tous les champs définis dans la table.**

## 7.2.3 Les critères de sélection

Les critères de sélection constituent une expression logique; qui peut prendre la valeur '**Vrai**' ou '**Faux**'. Les critères de sélection sont appliqués à chaque enregistrement d'une table. Lorsque pour un enregistrement donné, l'expression logique prend la valeur 'Vrai', cet enregistrement :

- fait partie du résultat pour une requête de sélection;
- est modifié pour une requête de modification;
- est effacé pour une requête de suppression;

**Comparaison à une valeur donnée.**

Pour chaque enregistrement, la valeur d'un champ donné est comparée à une valeur fixe. Cette valeur fixe est généralement une valeur numérique, une date ou un texte.

Voici les opérateurs de comparaison:

| | |
|---|---|
| = | "est égal" |
| > | "strictement supérieur" |
| < | "strictement inférieur" |
| >= | "supérieur ou égal" |
| <= | "inférieur ou égal" |
| <> | "est différent" |

Exemples:

1. Afficher le prénom et le nom de tous les employés du service "Marketing"

```
SELECT fldPrénom, fldNom
FROM tblEmployés
WHERE fldService='Marketing';
```

2. Afficher le prénom, le nom et l'âge de tous les employés plus jeunes que 50 ans

```
SELECT fldPrénom, fldNom, fldAge
FROM tblEmployés
WHERE fldAge<50;
```

Quel problème se pose lorsqu'on exécute cette même requête encore une fois un an plus tard ?

Comment peut-on éviter un tel problème dès le départ, déjà lors de la conception des tables ?

3. Augmentez de la valeur 1 l'âge de Madame Angela Portante.

```
UPDATE tblEmployés
SET fldAge=fldAge+1
WHERE fldNom='Portante';
```

**Remarque:**

Cette requête peut provoquer des résultats imprévus au cas ou plusieurs employés ont par exemple le même nom. Pour être certain de ne pas commettre d'erreur, il faudrait d'abord sélectionner tous les employés qui s'appellent "Portante". Lorsque cette requête ne fournit qu'un seul enregistrement, vous pouvez exécuter la requête comme indiqué en haut. Par contre lorsque vous remarquez que la table contient plusieurs employés au nom de "Portante", vérifiez les enregistrements, et retenez la valeur de la clé primaire (*idEmployé*) pour l'employé que vous désirez modifier. Ensuite utilisez la valeur de *idEmployé* dans la partie WHERE de la commande UPDATE (….WHERE idEmployé=<valeur>).

4. Effacez tous les employés du service Informatique.

```
DELETE FROM tblEmployés
WHERE fldService='Informatique';
```

## 7.2.4 Comparaison à un filtre

Parfois, on ne connaît pas la valeur exacte à laquelle on veut comparer la valeur d'un champ. Dans ce cas on peut utiliser un filtre. Un filtre est une expression qui peut contenir des lettres, des chiffres et en plus les 2 caractères spéciaux (angl. Wildcards) suivants:

- **%** représente n'importe quelle séquence de 0 ou plusieurs caractères;
- **_** représente un seul caractère quelconque.

Exemple: Pour rechercher des personnes dont le nom est 'SCHMITZ' ou 'SCHMITT' ou 'SCHMIT' etc. on définit par exemple le filtre suivant : **'SCHMI%'**

Exemple: Le filtre **'BL__'** sélectionne par exemple les valeurs 'BLEU' ou 'BLUE' mais pas 'BLANC'

Les filtres sont utilisés ensemble avec le mot réservé LIKE. Voici la syntaxe:

. . .
**WHERE <Nom du champ> LIKE <Filtre>**

Exemples:

1. Afficher le nom et le prénom des employés dont le prénom contient un trait d'union (p.ex. Jean-Jacques)

```
SELECT fldNom, fldPrénom
FROM tblEmployés
WHERE fldPrénom LIKE '%-%';
```

2. Afficher le nom, le prénom et l'âge des employés dont le nom commence par 'W', est composé de 5 lettres et se termine par 'R'

```
SELECT fldNom, fldPrénom, fldAge
FROM tblEmployés
WHERE fldNom LIKE 'W___R';
```

**Remarque**

Pour les manipulations pratiques, il faut se rendre compte que certains SGBD utilisent des caractères spéciaux différents pour représenter une séquence de caractères respectivement un caractère quelconque. MS-Access par exemple utilise les caractères suivants:

| | SQL | MS-Access |
|---|---|---|
| **Séquence de 0 ou plusieurs caractères** | **%** | ***** |
| **Un seul caractère quelconque** | **_** | **?** |

## 7.2.5 Les opérateurs logiques

 Il existe 3 opérateurs logiques:

1. **NOT** (Négation logique)

   L'opérateur NOT inverse le résultat d'une expression logique.

2. **AND** (Et logique)

   L'opérateur AND nous permet de combiner plusieurs conditions dans une expression logique. L'expression logique retourne uniquement la valeur 'Vrai' lorsque toutes les conditions sont remplies.

3. **OR** (Ou logique)

   L'opérateur OR nous permet de combiner plusieurs conditions dans une expression logique. L'expression logique retourne la valeur 'Vrai' lorsque au moins une des conditions est remplie.

**Priorité des opérateurs logiques**

Lorsqu'on combine plusieurs conditions par des opérateurs logiques, le résultat final de l'expression logique dépend de l'ordre d'exécution des différentes conditions. Cet ordre est déterminé par la priorité des opérateurs logiques. Voici l'ordre prédéfini en SQL:

1. Déterminer le résultat logique ('Vrai','Faux') des comparaisons (=, <, > etc.)
2. Effectuer les négations (NOT)
3. Effectuer les AND
4. Effectuer les OR

Pour modifier cet ordre d'exécution, nous pouvons utiliser des parenthèses afin de grouper les différentes conditions logiques.

Exemples

1. Afficher le prénom et le nom de tous les employés qui ne travaillent pas dans le service "Marketing"

```
SELECT fldPrénom, fldNom
FROM tblEmployés
WHERE NOT fldService='Marketing';
```

   Formulez une requête qui affiche exactement le même résultat, sans utiliser l'opérateur NOT.

```
SELECT fldPrénom, fldNom
```

```
FROM tblEmployés
WHERE fldService<>'Marketing';
```

2. Afficher le numéro d'employé, le prénom et le nom de tous les employés dont le nom ne commence pas par la lettre 'W'

```
SELECT idEmployé, fldPrénom, fldNom
FROM tblEmployés
WHERE fldNom NOT LIKE 'W%';
```

3. Afficher le numéro de l'employé, le prénom et le nom pour les employés du service Informatique qui ont moins de 30 ans.

```
SELECT idEmployé, fldPrénom, fldNom
FROM tblEmployés
WHERE fldService='Informatique' AND fldAge<30;
```

4. Afficher le prénom et nom des employés féminins (code=F) qui ne travaillent pas au service marketing.

```
SELECT fldPrénom, fldNom
FROM tblEmployés
WHERE fldSexe='F' AND NOT fldService='Marketing';
```

ou

```
SELECT fldPrénom, fldNom
FROM tblEmployés
WHERE fldSexe='F' AND fldService<>'Marketing';
```

5. Afficher tous les champs pour les employés de nationalité luxembourgeoise (Code=LUX) ou portugaise (Code=PRT).

```
SELECT *
FROM tblEmployés
WHERE fldNationalité='LUX' OR fldNationalité='PRT';
```

6. L'employé Emil Meier est transféré du service Comptabilité dans le service Informatique. Reflétez ce changement dans la table.

```
UPDATE tblEmployés
SET fldService='Informatique'
WHERE fldPrénom='Emil' AND fldNom='Meier';
```

**Remarque:**

Cette requête peut provoquer des résultats imprévus au cas ou plusieurs employés ont par exemple le même nom. Pour être certain de ne pas commettre d'erreur, il faudrait d'abord sélectionner tous les employés qui s'appellent Emil Meier. Lorsque cette requête ne fournit qu'un seul enregistrement, vous pouvez exécuter la requête comme indiqué en haut. Par contre lorsque vous remarquez que la table contient plusieurs employés au nom de Emil Meier, vérifiez les enregistrements, et retenez la valeur de la clé primaire (idEmployé) pour l'employé que vous désirez modifier. Ensuite utilisez la valeur de idEmployé dans la partie WHERE de la commande UPDATE (….WHERE idEmployé=<valeur>).

7. Affichez tous les champs pour les employés féminins de nationalité luxembourgeoise (Code='LUX') ou allemande (Code='ALL')

```
SELECT *
FROM tblEmployés
WHERE (fldNationalité='LUX' OR fldNationalité='ALL') AND fldSexe='F';
```

Est-ce que cette requête serait correcte sans les parenthèses ?

## 7.2.6 Valeur zéro, chaîne vide et valeur indéterminée (NULL)


Généralement, chaque champ défini dans une table possède une valeur bien définie. Il existe pourtant des situations dans lesquelles cette valeur est inconnue ou n'existe pas temporairement.
La quantité en stock d'un nouveau produit par exemple peut être 0, le prénom ne s'applique pas du tout pour un client du type "Société" tandis que l'adresse d'un nouveau client peut être encore inconnue lors de l'insertion des données du client dans une BD.

Les SGBD nous offrent en général 3 valeurs pour ces types de situations:
- Le nombre 0 ;
- La chaîne de caractères vide (") ;
- La valeur prédéfinie **NULL** (Valeur indéterminée, Champ vide).

Il n'est pas toujours évident de décider sur la bonne valeur. Voici quelques réflexions concernant les exemples énoncés:
- Pour le stock d'un produit, qui est 0 au début, il est conseillé d'insérer effectivement dès le début la valeur numérique 0. On a effectivement 0 produits dans le stock ce qui ne veut pas dire que la quantité en stock est indéterminée.
- Pour le prénom d'un client du type "Société" on utilise la chaîne vide ("). Le prénom d'une société est définitivement non-existant.
- En ce qui concerne l'adresse d'un nouveau client, on insère la valeur NULL, ce qui veut dire que l'adresse est (pour le moment) indéterminée. On est plus ou moins sûr de connaître l'adresse à un moment ultérieur.
- Lorsque pour un nouveau client, vous ne connaissez pas le numéro de fax, on doit distinguer deux cas. Si le client n'a pas de fax, vous n'aurez définitivement pas son numéro de fax, et vous pouvez affecter la chaîne vide au champ. Si par contre le client possède un numéro de fax, mais pour une raison ou l'autre vous l'ignorez encore, vous devez affecter la valeur NULL au champ. En cas de doute, affectez la valeur NULL.

**En général, on peut dire que la valeur NULL est uniquement affectée à un champ en cas d'indétermination de la valeur du champ.**

Remarques:

On peut insérer la valeur NULL de façon explicite, par exemple à l'aide d'une requête d'insertion. La plupart des SGBD insèrent automatiquement la valeur NULL pour chaque champ qui n'a pas de valeur explicite associée dans une requête d'insertion.

Chaque expression logique, qui contient une condition indéterminée prend la valeur NULL. On peut donc affirmer, qu'un enregistrement qui contient une valeur NULL pour un champ donné, n'apparaît pas dans une sélection portant sur une comparaison avec une valeur pour ce champ.

L'opérateur **IS NULL** nous permet de tester de façon explicite si une valeur est indéterminée pour un champ. L'opérateur **IS NOT NULL** permet de tester inversement le fait que la valeur est bien déterminée.

Exemple:

1. Vous devez ajouter un nouvel employé dans la BD. Voici les informations dont vous disposez:

| | | | | | | |
|---|---|---|---|---|---|---|
| 24 | Marcel | Schrobiltgen | LUX | **?** | M | **?** |

Sachant que M.Schrobiltgen n'est pas du tout affecté à un service spécifique, puisqu'il est le réviseur interne de l'entreprise, formulez la requête d'insertion.

```
INSERT INTO tblEmployés
VALUES (24, 'Marcel', 'Schrobiltgen', 'LUX', NULL, 'M', '');
```

ou

```
INSERT INTO tblEmployés (idEmployé, fldPrénom, fldNom, fldNationalité, fldSexe, fldService)
VALUES (24, 'Marcel', 'Schrobiltgen', 'LUX', 'M', '');
```

Attention: On ne connaît pas encore l'âge de M.Schrobiltgen, puisqu'on a probablement oublié de le lui demander. Toutefois on est sûr de le connaître plus tard, ce qui veut dire que pour l'instant son âge est indéterminé et qu'on affecte la valeur NULL au champ *fldAge*. En ce qui concerne le service, celui-ci n'est pas vraiment indéterminé, puisqu'on sait très bien que M.Schrobiltgen n'est pas du tout affecté à un service. Pour cela , le champ *fldService* contiendra une chaîne vide.

**Exercice**

Dans laquelle des requêtes suivantes va apparaître M.Schrobiltgen ?

```
SELECT *
FROM tblEmployés
WHERE fldAge=0;

SELECT *
FROM tblEmployés
WHERE fldAge IS NULL;

SELECT *
FROM tblEmployés
WHERE fldService='';

SELECT *
FROM tblEmployés
WHERE fldService IS NULL;

SELECT *
FROM tblEmployés
WHERE fldAge IS NOT NULL AND fldService='';

SELECT *
FROM tblEmployés
WHERE fldAge IS NOT NULL OR fldService='';
```

## 7.2.7 Comparaison à une fourchette de valeurs

 L'opérateur **BETWEEN ... AND ...** permet de déterminer si la valeur d'un champ donné appartient à un intervalle bien défini. L'intervalle est généralement un intervalle numérique ou un intervalle du type Date.

Exemples

1. Afficher le numéro d'employé, le nom et l'âge des employés âgés entre 30 et 50 ans.

```
SELECT idEmployé, fldNom, fldAge
FROM tblEmployés
WHERE fldAge BETWEEN 30 AND 50;
```

   Formulez une requête qui affiche exactement le même résultat, sans utiliser l'opérateur BETWEEN ... AND ....

```
SELECT idEmployé, fldNom, fldAge
FROM tblEmployés
WHERE fldAge>=30 AND fldAge<=50;
```

2. Afficher tous les champs pour les employés masculins âgés entre 20 et 30 ans et les employés féminins âgés entre 40 et 50 ans.

```
SELECT *
FROM tblEmployés
WHERE fldSexe='M' AND fldAge BETWEEN 20 AND 30 OR fldSexe='F' AND fldAge BETWEEN 40
       AND 50;
```

## 7.2.8 Comparaison à une liste de valeurs

L'opérateur **IN (<Liste de valeurs>)** permet de déterminer si la valeur d'un champ donné appartient à une liste de valeurs prédéfinies.

Les valeurs dans la liste des valeurs sont généralement des valeurs numériques, des valeurs du type Texte ou des valeurs du type Date.

Exemples:

1. Afficher le numéro d'employé, le nom, l'âge et le service des employés qui sont affectés aux services 'Comptabilité', 'Informatique' et 'Vente'.

```
SELECT idEmployé, fldNom, fldAge, fldService
FROM tblEmployés
WHERE fldService IN ('Comptabilité','Informatique','Vente');
```

   Formulez une requête qui affiche exactement le même résultat, sans utiliser l'opérateur IN.

```
SELECT idEmployé, fldNom, fldAge, fldService
FROM tblEmployés
WHERE fldService='Comptabilité' OR fldService='Informatique'OR fldService='Vente';
```

2. Afficher tous les champs pour les employés masculins, âgés d'au moins 30 ans qui ne sont pas de nationalité luxembourgeoise (Code='LUX'), portugaise (Code='PRT'), allemande (Code='ALL') ou italienne (Code='ITA')

```
SELECT *
FROM tblEmployés
WHERE fldSexe='M' AND fldAge>=30 AND fldNationalité NOT IN ('LUX', 'PRT', 'ALL',
       'ITA');
```

## 7.2.9 Définir l'ordre d'une requête de sélection

L'ordre obtenu dans la réponse d'une requête de sélection a été laissé jusqu'à maintenant au pur hasard.

 L'expression ORDER BY nous permet de définir convenablement l'ordre d'apparition des enregistrements qui vérifient les critères de sélection de la requête. Voici la syntaxe:

```
SELECT <Nom des champs>
FROM <Nom de la table>
WHERE <Critères de sélection>
ORDER BY <Nom d'un champ>[ASC/DESC], <Nom d'un champ>[ASC/DESC], ...  ;
```

Par défaut l'ordre de tri est ascendant (ASC), donc vous n'avez pas nécessairement besoin d'indiquer le mot ASC. Cependant, lorsque vous voulez trier les enregistrements en ordre descendant, le mot DESC est indispensable.

**Exemples:**

Soit la table suivante.

**tblLivres**

| Nom du champ | Type de données | Description |
|---|---|---|
| idLivre | Numérique | Numéro du livre (Clé primaire) |
| fldTitre | Texte | Titre du livre |
| fldAuteur | Texte | Auteur du livre |
| fldLangue | Texte | Langue (ALL, FRA, ANG) |
| fldGenre | Texte | Genre du livre (Roman, Technique, Histoire …) |
| fldPrix | Numérique | Prix de vente du livre |
| fldEnStock | Numérique | Quantité d'exemplaires en stock |

Exemple 1:

```
SELECT idLivre, fldTitre, fldAuteur, fldPrix
FROM tblLivres
ORDER BY fldPrix;
```

respectivement

```
SELECT idLivre, fldTitre, fldAuteur, fldPrix
FROM tblLivres
ORDER BY fldPrix ASC;
```

Query9 : Select Query

| idLivre | fldTitre | fldAuteur | fldPrix |
|---|---|---|---|
| 99832 | Dracula | Bram Stoker | 450 |
| 87777 | Roter Drache | Thomas Harris | 489 |
| 38366 | Die Prüfung | F.Paul Wilson | 600 |
| 57296 | Le micro ... comment ça marche ? | Ron White | 824 |
| 33344 | Teach yourself Java in 21 days | Charles Perkins | 1065 |
| 78999 | Der letzte Zar | Klaus Werheim | 1074 |
| 87644 | Novell Netware 4.1 | Pierre Godefroid | 1138 |
| 34000 | MS-Access 2.0 | Ken Getz | 1377 |
| 98222 | Der Zerfall des Sowjetimperium | Alexeji Kolimov | 1436 |
| 0 | | | 0 |

Record: 1 of 9

Exemple 2:

```
SELECT idLivre, fldTitre, fldAuteur, fldPrix
FROM tblLivres
ORDER BY fldPrix DESC;
```

Query9 : Select Query

| idLivre | fldTitre | fldAuteur | fldPrix |
|---|---|---|---|
| 98222 | Der Zerfall des Sowjetimperium | Alexeji Kolimov | 1436 |
| 34000 | MS-Access 2.0 | Ken Getz | 1377 |
| 87644 | Novell Netware 4.1 | Pierre Godefroid | 1138 |
| 78999 | Der letzte Zar | Klaus Werheim | 1074 |
| 33344 | Teach yourself Java in 21 days | Charles Perkins | 1065 |
| 57296 | Le micro ... comment ça marche ? | Ron White | 824 |
| 38366 | Die Prüfung | F.Paul Wilson | 600 |
| 87777 | Roter Drache | Thomas Harris | 489 |
| 99832 | Dracula | Bram Stoker | 450 |
| 0 | | | 0 |

Record: 1 of 9

On peut aussi trier sur plusieurs champs. Pour afficher tous les livres triés d'abord sur leur genre en ordre ascendant et pour chaque genre sur le prix en ordre descendant, on utilise la requête suivante:

Exemple 3:

```
SELECT idLivre, fldTitre, fldAuteur, fldGenre, fldPrix
FROM tblLivres
ORDER BY fldGenre ASC, fldPrix DESC;
```

Query9 : Select Query

| idLivre | fldTitre | fldAuteur | fldGenre | fldPrix |
|---|---|---|---|---|
| 98222 | Der Zerfall des Sowjetimperium | Alexeji Kolimov | Histoire | 1436 |
| 78999 | Der letzte Zar | Klaus Werheim | Histoire | 1074 |
| 38366 | Die Prüfung | F.Paul Wilson | Roman | 600 |
| 87777 | Roter Drache | Thomas Harris | Roman | 489 |
| 99832 | Dracula | Bram Stoker | Roman | 450 |
| 34000 | MS-Access 2.0 | Ken Getz | Technique | 1377 |
| 87644 | Novell Netware 4.1 | Pierre Godefroid | Technique | 1138 |
| 33344 | Teach yourself Java in 21 days | Charles Perkins | Technique | 1065 |
| 57296 | Le micro ... comment ça marche ? | Ron White | Technique | 824 |
| 0 | | | | 0 |

Record: 1 of 9

Nous remarquons que **l'ordre de tri est basé sur l'ordre alphabétique pour les champs de type TEXTE et sur l'ordre numérique pour les champs de type NUMERIQUE**. La plupart des SGBD sont également capable de trier des valeurs de type DATE.

Exemple 4:

Afficher le numéro du livre, le titre, l'auteur et la langue de tous les romans. Triez la liste en ordre descendant sur la langue.

```
SELECT idLivre, fldTitre, fldAuteur, fldLangue
FROM tblLivres
WHERE fldGenre='Roman'
ORDER BY fldLangue DESC;
```

## 7.2.10 Les valeurs calculées

Dans une requête on a la possibilité de définir des champs à valeur calculée. Un tel champ ne fait pas partie d'une table, mais contient une valeur, qui est calculée sur base d'un ou de plusieurs champs existants.

Exemple:

```
SELECT idLivre, fldTitre, fldPrix*1.15 AS PrixTTC
FROM tblLivres;
```

Champ à valeur calculée

Query13 : Select Query

| idLivre | fldTitre | PrixTTC |
|---|---|---|
| 33344 | Teach yourself Java in 21 days | 1224,75 |
| 34000 | MS-Access 2.0 | 1583,55 |
| 38366 | Die Prüfung | 690 |
| 57296 | Le micro ... comment ça marche ? | 947,6 |
| 78654 | L'homme juste | 281,75 |
| 78999 | Der letzte Zar | 1235,1 |
| 87644 | Novell Netware 4.1 | 1308,7 |
| 87777 | Roter Drache | 562,35 |
| 98222 | Der Zerfall des Sowetimperiums | 1651,4 |
| 99832 | Dracula | 517,5 |

Record: 1 of 10

Remarque: On peut utiliser un champ à valeur calculée pour renommer un champ affiché dans une requête.

Exemple:

```
SELECT idLivre As ISBN , fldTitre
FROM tblLivres;
```

Champ à valeur calculée

Query13 : Select Query

| ISBN | fldTitre |
|---|---|
| 33344 | Teach yourself Java in 21 days |
| 34000 | MS-Access 2.0 |
| 38366 | Die Prüfung |
| 57296 | Le micro ... comment ça marche ? |
| 78654 | L'homme juste |
| 78999 | Der letzte Zar |
| 87644 | Novell Netware 4.1 |
| 87777 | Roter Drache |
| 98222 | Der Zerfall des Sowetimperiums |
| 99832 | Dracula |
| 0 | |

Record: 1 of 10

Un champ à valeur calculée n'est pas à confondre aux calculs qui peuvent intervenir à l'intérieur d'un critère de sélection. Comme un critère de sélection n'est rien d'autre qu'une expression, qui peut être évaluée soit à la valeur logique VRAI, soit à la valeur logique FAUX, la requête suivante est absolument correcte, mais ne définit pas de champ à valeur calculée.

```
SELECT idLivre, fldTitre
FROM tblLivres
```

```
WHERE (fldPrix*fldEnStock)<5000;
```

## 7.2.11 Les fonctions d'agrégation

Derrière ce mot compliqué se cachent quelques fonctions qui peuvent être utilisées à l'intérieur des requêtes de sélection pour faire des calculs sur le résultat de la requête. Imaginons la requête suivante:

```
SELECT *
FROM tblLivres
WHERE fldEnStock=0;
```

Cette requête nous retourne tous les livres dont il n'y a plus d'exemplaire en stock. Il se peut très bien que l'utilisateur ne soit pas intéressé dans le détail, mais veut uniquement connaître le nombre de livres dont il n'y a plus d'exemplaires en stock. La requête correspondante est:

```
SELECT COUNT (*)
FROM tblLivres
WHERE fldEnStock=0;
```

Le résultat de cette requête est une **valeur unique** indiquant combien de livres se trouvent dans la table avec le champ *fldEnStock* ayant la valeur 0.

**Les paramètres d'une fonction d'agrégation sont toujours entourés de parenthèses.**

Voici les fonctions d'agrégations les plus répandues:

| | |
|---|---|
| **COUNT (*)** | Détermine le nombre d'enregistrements du résultat de la requête. Tient compte de tous les enregistrements, y inclus ceux contenant des valeurs NULL |
| **COUNT (Nom d'un champ)** | Détermine le nombre des enregistrements pour lesquels le champ indiqué ne contient pas la valeur NULL |
| **SUM (Nom d'un champ)** | Calcule pour tous les enregistrements sélectionnés, la somme des valeurs du champ indiqué, pourvu que cette valeur soit différente de NULL. |
| **AVG (Nom d'un champ)** | Calcule pour tous les enregistrements sélectionnés, la moyenne des valeurs du champ indiqué, pourvu que cette valeur soit différente de NULL. |
| **MAX (Nom d'un champ)** | Détermine pour tous les enregistrements sélectionnés, la plus grande des valeurs du champ indiqué, pourvu que cette valeur soit différente de NULL. |
| **MIN (Nom d'un champ)** | Détermine pour tous les enregistrements sélectionnés, la plus petite des valeurs du champ indiqué, pourvu que cette valeur soit différente de NULL. |

Exemples:

1. Affichez la moyenne des prix des livres allemands.

```
SELECT AVG(fldPrix)
FROM tblLivres
WHERE fldLangue='ALL';
```

2. Affichez la quantité totale des exemplaires de romans allemands en stock.

```
SELECT SUM(fldEnStock)
FROM tblLivres
WHERE fldGenre='Roman' AND fldLangue='ALL';
```

3. Combien de livres d'histoires sont disponibles en langue française ?

```
SELECT COUNT(*)
FROM tblLivres
WHERE fld Genre='Histoire' AND fldLangue='FRA';
```

4. Déterminez le prix du roman anglais le plus cher qui est actuellement disponible.

```
SELECT MAX(fldPrix)
FROM tblLivres
WHERE fldGenre='Roman' AND fldLangue='ANG' AND fldEnStock>0;
```

**Remarque:**

Les fonctions d'agrégation admettent comme paramètre également des expressions contenant plusieurs champs ainsi que DISTINCT.

Exemples:

1. Calculez la valeur actuelle du stock

```
SELECT SUM(fldPrix * fldEnStock)
FROM tblLivres;
```

2. Combien de langues différentes sont représentées dans notre stock de livres

```
SELECT COUNT(DISTINCT fldLangue)
FROM tblLivres;
```

**A faire : Exercice 1**

## 7.2.12 Requêtes sur les groupes

### *7.2.12.1 La clause GROUP BY*

Reprenons notre table avec les livres d'une librairie.

**tblLivres**

| Nom du champ | Type de données | Description |
|---|---|---|
| idLivre | Numérique | Numéro du livre (Clé primaire) |
| fldTitre | Texte | Titre du livre |
| fldAuteur | Texte | Auteur du livre |
| fldLangue | Texte | Langue (ALL, FRA, ANG) |
| fldGenre | Texte | Genre du livre (Roman, Technique, Histoire ...) |
| fldPrix | Numérique | Prix de vente du livre |
| fldEnStock | Numérique | Quantité d'exemplaires en stock |

Soit la requête suivante:

```
SELECT fldTitre,fldGenre,fldEnStock
FROM tblLivres;
```

Voici à titre d'exemple le résultat de cette requête

Query35 : Select Query

| fldTitre | fldGenre | fldEnStock |
|---|---|---|
| Teach yourself Java in 21 days | Technique | 13 |
| MS-Access 2.0 | Technique | 5 |
| Ms-Access 97 | Technique | 0 |
| New Windows 98 | Technique | 23 |
| Die Prüfung | Roman | 3 |
| Le micro ... comment ça marche ? | Technique | 2 |
| L'homme juste | Roman | 3 |
| Der letzte Zar | Histoire | 2 |
| Novell Netware 4.1 | Technique | 3 |
| Roter Drache | Roman | 3 |
| Der Zerfall des Sowetimperiums | Histoire | 2 |
| Dracula | Roman | 3 |

Record: 1 of 12

Si on voulait connaître la quantité en stock par genre de livre, on aurait le résultat suivant:

Query35 : Select Query

| fldGenre | 'En Stock' |
|---|---|
| Histoire | 4 |
| Roman | 12 |
| Technique | 46 |

Record: 1

SQL nous offre une extension à la requête de sélection, qui nous permet de formuler exactement ce type de questions. La clause **GROUP BY <Liste des champs de groupe>** répartit le résultat d'une requête de sélection en groupes.

Généralement, on applique une fonction d'agrégation aux membres de chaque groupe.

Voici la requête qui nous donne le tableau précédant:

```
SELECT fldGenre, SUM(fldEnStock) As 'En Stock'
FROM tblLivres
GROUP BY fldGenre;
```

La clause **GROUP BY fldGenre** crée des groupes selon les valeurs du champ *fldGenre*, c.à.d. les 3 groupes 'Histoire' , 'Roman' et 'Technique'.

La partie **SELECT fldGenre, SUM(fldEnStock)** affiche pour chaque groupe une seule ligne, qui contient la valeur du champ de groupe *fldGenre*, ainsi que la somme des valeurs du champ *fldEnStock*.

La partie ... **AS 'En Stock'** est uniquement utilisée afin de renommer l'en-tête du champ calculé via la fonction d'agrégation SUM.

**La clause GROUP BY**

La clause **GROUP BY <Liste des champs de groupe>** intervient sur le résultat d'un SELECT. En fait, les enregistrements résultant d'une requête de sélection sont groupés, de façon qu'à l'intérieur de chaque groupe, les valeurs pour la liste des champs de groupe soient identiques.

Généralement, on applique une fonction d'agrégation à un ou plusieurs champs, ne faisant pas partie de la liste des champs de groupe.

**Attention:** La clause SELECT peut uniquement contenir des champs faisant partie de la liste des champs de groupe ou des fonctions d'agrégation appliquées à un des autres champs.

La requête de sélection peut bien sûr contenir des critères de sélection (WHERE ...), qui éliminent un certain nombre d'enregistrements déjà avant la création des groupes.

On a la possibilité d'appliquer la clause ORDER BY au résultat d'un GROUP BY

**Syntaxe:**

```
SELECT <Liste des champs de groupe>,[COUNT/SUM...]
FROM <Nom de la table>
WHERE <Critères de sélection>
GROUP BY <Liste de champs de groupe>
ORDER BY <Nom d'un champ>[ASC/DESC], <Nom d'un champ>[ASC/DESC], ...;
```

Exemples:

1. Classez les genres de livres par ordre descendant et affichez pour chaque genre la moyenne du prix.

```
SELECT fldGenre, AVG(fldPrix)
FROM tblLivres
GROUP BY fldGenre
ORDER BY fldGenre DESC;
```

2. Même question, mais en tenant uniquement compte des livres anglais

```
SELECT fldGenre, AVG(fldPrix)
FROM tblLivres
WHERE fldLangue='ANG'
GROUP BY fldGenre
ORDER BY fldGenre DESC;
```

3. Affichez pour chaque genre, le nombre de titres disponibles, ainsi que la quantité d'exemplaires en stock.

```
SELECT fldGenre, COUNT(fldTitre), SUM(fldEnStock)
FROM tblLivres
GROUP BY fldGenre;
```

ou

```
SELECT fldGenre, COUNT(*), SUM(fldEnStock)
FROM tblLivres
GROUP BY fldGenre;
```

4. Regroupez les livres par genre et par langue et affichez pour chaque groupe la valeur en stock des livres. Triez le résultat par ordre ascendant sur les langues, et à l'intérieur d'une langue par ordre ascendant sur le genre.

```
SELECT fldGenre, fldlangue, SUM(fldPrix*fldEnStock)
FROM tblLivres
GROUP BY fldGenre, fldLangue
ORDER BY fldLangue, fldGenre;
```

**Exercice**

Comparez les deux requêtes suivantes.

| | |
|---|---|
| `SELECT fldLangue`<br>`FROM tblLivres`<br>`GROUP BY fldLangue;` | `SELECT DISTINCT fldLangue`<br>`FROM tblLivres;` |

**Remarque:**
Si pour un champ de groupe, les valeurs d'un ou de plusieurs enregistrements sont indéterminées (NULL), alors ces enregistrements sont regroupées dans un groupe séparé (Groupe 'NULL').

### 7.2.12.2 La clause HAVING

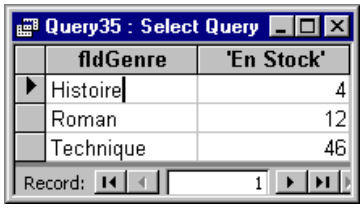
Sachant que les critères de sélection (WHERE ...) nous permettent d'éliminer un certain nombre d'enregistrements avant la création des groupes, il serait intéressant de disposer d'une deuxième possibilité de filtrage, qui s'applique aux groupes eux-mêmes.

La clause **HAVING <Critères de sélection des groupes>** nous offre la possibilité d'éliminer du résultat; les groupes qui ne donnent pas satisfaction aux critères de sélection des groupes.

Reprenons l'exemple:

```
SELECT fldGenre, SUM(fldEnStock) AS 'En Stock'
FROM tblLivres
GROUP BY fldGenre;
```

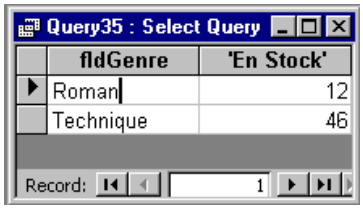
qui nous donne le résultat suivant:




Lorsqu'on veut par exemple uniquement afficher les groupes pour lesquelles la quantité en stock est supérieure à 10, on utilise la clause HAVING de la façon suivante:

```
SELECT fldGenre, SUM(fldEnStock) AS 'En Stock'
FROM tblLivres
GROUP BY fldGenre
HAVING SUM(fldEnStock)>10;
```

Voici le résultat correspondant.




Est-ce que la requête suivante donne le même résultat ? Expliquez.

```
SELECT fldGenre, SUM(fldEnStock) AS 'En Stock'
FROM tblLivres
WHERE fldEnStock>10
GROUP BY fldGenre;
```

**La clause HAVING**

La clause **HAVING <Critères de sélection des groupes>** est uniquement spécifiée en relation avec un GROUP BY. Une fois les groupes créés, cette clause en élimine certains, basés sur les critères de sélection des groupes.

Les critères de sélection des groupes portent bien entendu sur la valeur d'une ou de plusieurs des fonctions d'agrégation calculées pour chaque groupe.

**Syntaxe:**

```
SELECT <Liste des champs de groupe>,[COUNT/SUM...]
FROM <Nom de la table>
WHERE <Critères de sélection>
GROUP BY <Liste de champs de groupe>
HAVING <Critères de sélection des groupes>
ORDER BY <Nom d'un champ>[ASC/DESC], <Nom d'un champ>[ASC/DESC], ...;
```

Exemples:

1. Affichez pour chaque langue, le nombre de titres disponibles, ainsi que la quantité d'exemplaires en stock, en tenant uniquement compte des langues pour lesquelles la quantité d'exemplaires est supérieure à 0.

```
SELECT fldLangue, COUNT(fldTitre), SUM(fldEnStock)
FROM tblLivres
GROUP BY fldLangue
HAVING SUM(fldEnStock)>0;
```

ou

```
SELECT fldLangue, COUNT(*), SUM(fldEnStock)
FROM tblLivres
GROUP BY fldLangue
HAVING SUM(fldEnStock)>0;
```

2. Même question, mais en éliminant dès le début les livres anglais

```
SELECT fldLangue, COUNT(fldTitre), SUM(fldEnStock)
FROM tblLivres
WHERE fldLangue<>'ANG'
GROUP BY fldLangue
HAVING SUM(fldEnStock)>0;
```

ou

```
SELECT fldLangue, COUNT(*), SUM(fldEnStock)
FROM tblLivres
WHERE fldLangue<>'ANG'
GROUP BY fldLangue
HAVING SUM(fldEnStock)>0;
```

3. Classez les auteurs par ordre descendant en fonction du nombre de titres. Tenez uniquement compte des titres français et allemands, et des auteurs ayant au moins 3 titres disponibles.

```
SELECT fldAuteur, COUNT(fldTitre)
FROM tblLivres
WHERE fldLangue IN ('FRA', 'ALL')
GROUP BY fldAuteur
HAVING COUNT(fldTitre)>=3
ORDER BY COUNT(fldTitre) DESC;
```

ou

```
SELECT fldAuteur, COUNT(*)
FROM tblLivres
WHERE fldLangue IN ('FRA', 'ALL')
GROUP BY fldAuteur
HAVING COUNT(*)>=3
ORDER BY COUNT(*) DESC;
```

**A faire : Exercice 2 et Exercice 3**

## 7.2.13 Exercices

### Exercice 1: Gestion de livres

Soit la table suivante pour stocker les livres d'une librairie:

**tblLivres**

| Nom du champ | Type de données | Description |
|---|---|---|
| idLivre | Numérique | Numéro du livre (Clé primaire) |
| fldTitre | Texte | Titre du livre |
| fldAuteur | Texte | Auteur du livre |
| fldLangue | Texte | Langue (ALL, FRA, ANG) |
| fldGenre | Texte | Genre du livre (Roman, Technique, Histoire ...) |
| fldPrix | Numérique | Prix de vente du livre |
| fldEnStock | Numérique | Quantité d'exemplaires en stock |

**Trouvez le code SQL pour les requêtes suivantes:**

1. Affichez le numéro, le titre, l'auteur et la quantité en stock pour tous les romans.
2. Affichez tous les champs des romans allemands. Effectuez un classement par ordre ascendant sur le numéro du livre.
3. Affichez les différentes langues dans lesquelles sont rédigés les livres de la librairie.
4. Affichez le numéro du livre "Windows 95" de "Pierre Godefroid" en version française.
5. Supposons que ce numéro soit 38285, augmentez de 10 unités la quantité en stock de ce livre.
6. Insérez le nouveau livre suivant:

| | | | | | | |
|---|---|---|---|---|---|---|
| 34000 | MS-Access 2.0 | Ken Getz | ANG | Technique | 2300 | 2 |

7. Insérez le nouveau livre suivant:

| | | | | |
|---|---|---|---|---|
| 34001 | MS-Access 97 | Ken Getz | ANG | Technique |

   Quelle est la valeur des champs *fldPrix* et *fldEnStock* pour cet enregistrement ?

8. Tous les livres des genres Technique et Histoire subissent une hausse de prix de 10%. Représentez cette situation dans votre table.
9. Comptez le nombre de livres dont il reste moins de 3 exemplaires en stock.
10. Supprimez tous les romans anglais.

11. Affichez le numéro, le titre et l'auteur pour tous les livres dont la valeur en stock est supérieure à 10000. La valeur en stock est le prix d'un livre multiplié par la quantité en stock pour ce livre.

12. Ajoutez 15% au prix pour tous les livres techniques dont le titre contient le mot 'Windows 98'.

13. Afficher toutes les informations pour les romans français (Code='FRA') et les romans allemands (Code='ALL'). Utilisez uniquement des opérateurs logiques et des opérateurs de comparaison.

    Formulez une requête alternative qui fournit le même résultat.

14. Indiquez 2 requêtes différentes pour afficher le numéro du livre pour tous les livres anglais (Code='ANG') actuellement en stock, dont le prix varie entre 500 et 1000 Luf.

15. Afficher une liste qui contient toutes les langues dont il existe au moins un roman plus cher que 300 Luf.

16. Affichez le prix moyen des romans anglais.

# Exercice 2: Gestion des clients

Une société utilise la table suivante pour gérer ses clients.

**tblClients**

| Nom du champ | Type de données | Description |
|---|---|---|
| idClient | Numérique | Numéro du client |
| fldNom | Texte | Nom du client |
| fldPrénom | Texte | Prénom du client |
| fldSexe | Texte | Sexe du client. Valeurs possibles: 'F' et 'M' |
| fldAdresse | Texte | Rue et numéro |
| fldCP | Texte | Code postal |
| fldLocalité | Texte | Localité |
| fldNoTel | Texte | Numéro de téléphone |
| fldNoFax | Texte | Numéro de fax |
| fldDateNaiss | Date/Heure | Date de naissance du client |
| fldBonClient | Booléen (Logique) | Valeurs possibles: YES/NO |

**Trouvez le code SQL pour les requêtes suivantes:**

1. Affichez le nom, prénom, adresse, code postal et localité pour tous les clients habitant à Walferdange.

2. Insérez le client suivant dans la table:

| | | | | |
|---|---|---|---|---|
| 6 | Heinen | Edmond | M | 12, Am Gaard |

| | | | | | |
|---|---|---|---|---|---|
| 2021 | Ettelbruck | 217070 | 217071 | 12/01/1954 | NO |

   Indiquez la requête correcte pour ajouter ce client lorsque vous ignorez le numéro de fax.

   Indiquez la requête correcte pour ajouter ce client lorsque vous savez que le client ne possède pas de fax.

3. Affichez la liste de toutes les localités présentes dans la table des clients.

4. Tous les clients habitant à Ettelbruck, dans la Cité Patton, auront le nouveau code postal 8897. Remarque: Utilisez un filtre pour retrouver les adresses correctes.

5. Nous voulons "nettoyer" notre BD. En fait, il y a un certain nombre de clients dont l'adresse, le code postal, la localité le numéro de téléphone et le numéro de fax sont indéterminés. Ces enregistrements sont sans aucune valeur commerciale pour nous. Formulez une requête qui garde uniquement les clients pour lesquels on connaît:
   - soit le numéro de téléphone;
   - soit le numéro de fax;
   - soit l'adresse complète (*fldAdresse*, *fldCP*, *fldLocalité*).

Tous les autres clients sont effacés de la BD.

6. Comptez le nombre de clients masculins nés après le 1.1.1978.

7. Afficher le numéro client, le nom, le prénom, l'adresse, le code postal et la localité pour les bons clients féminins, à l'exception de ceux habitant à Luxembourg, Esch-s-Alzette et Ettelbruck.

8. Affichez par sexe, le nombre de clients nés après le 31/12/1969. Le champ qui affiche le nombre de clients doit porter l'en-tête 'Nombre de clients'.

9. Afficher le numéro client, le nom, le prénom, le sexe et la date de naissance pour les clients habitant à Luxembourg. Triez le résultat par ordre descendant sur le sexe et à l'intérieur, par ordre ascendant sur la date de naissance.

   Le premier enregistrement du résultat affiche donc les informations de la femme la plus jeune parmi les clients de Luxembourg. Est-ce que cette affirmation est correcte ?

---

10. Affichez le numéro client, le nom, le prénom et le code bon client pour tous les clients féminins habitant à Diekirch ou à Mersch. En ce qui concerne le code bon client, affichez l'en-tête 'Code spécial' au lieu d'afficher le nom du champ.

11. Affichez pour chaque localité le nombre de bons clients (cf. *fldBonClient*) ainsi que le nombre des autres clients. Triez la liste par ordre ascendant sur les localités, en affichant pour chaque localité d'abord le nombre de bons clients.

12. Déterminez la date de naissance du client le plus vieux habitant à Luxembourg.

# Exercice 3: Gestion de concerts

Une agence de concerts utilise la table suivante:

**tblConcerts**

| Nom du champ | Type de données | Description |
|---|---|---|
| idConcert | Numérique | Numéro d'identification (Clé primaire) |
| fldArtiste | Texte | Artiste ou groupe qui donne le concert |
| fldDate | Date/Heure | Date du concert |
| fldDébut | Date/Heure | Début du concert (Heure) |
| fldLocalité | Texte | Localité du concert |
| fldLieu | Texte | Lieu du concert |
| fldTypeLieu | Texte | (Centre Culturel , Hall Sportif , Club etc.) |
| fldPrixTicket | Numérique | Prix d'un ticket |
| fldPlaces | Numérique | Total des places disponibles pour le concert |
| fldTicketsVendus | Numérique | Nombre de tickets déjà vendus pour le concert |

Voici quelques exemples d'enregistrements:

tblConcerts : Table

| idConcert | fldArtiste | fldDate | fldDébut | fldLocalité | fldLieu | fldTypeLieu | fldPrixTicket | fldPlaces | fldTicketsVendus |
|---|---|---|---|---|---|---|---|---|---|
| 101 | Genesis | 20/04/1998 | 20:00 | Luxembourg | Stade Josy Weber | Terrain de Footbal | 1250 Luf | 9500 | 6765 |
| 102 | Massive Attack | 02/06/1998 | 21:00 | Luxembourg | Den Atelier | Club | 750 Luf | 1100 | 765 |
| 103 | Kelly Family | 13/07/1998 | 19:00 | Hupperdange | D'Scheier | Club | 500 Luf | 450 | 449 |
| 104 | Karel Gott | 15/07/1998 | 21:00 | Mondorf | Casino 500 | Autre | 1200 Luf | 700 | 700 |
| 105 | Bush | 16/07/1998 | 21:30 | Pétange | Centre Sportif | Hall Sportif | 800 Luf | 2200 | 1290 |
| 106 | Marianne & Michael | 20/07/1998 | 20:00 | Dudelange | Centre Sportif | Hall Sportif | 1000 Luf | 1900 | 1900 |
| 107 | Trei Schwäin | 22/07/1998 | 21:30 | Diekirch | Fête sous tente | Autre | 350 Luf | 1800 | 280 |
| 108 | Deep Purple | 02/08/1998 | 20:00 | Bascharage | Hall 75 | Centre Culturel | 1300 Luf | 1400 | 1097 |
| 109 | Life of Agony / Tiamat | 05/08/1998 | 20:30 | Luxembourg | Den Atelier | Club | 900 Luf | 1100 | 360 |
| 110 | Joe Cocker | 12/08/1998 | 20:00 | Pétange | Centre Sportif | Hall Sportif | 1100 Luf | 2200 | 2200 |
| 111 | Pulp | 20/08/1998 | 21:00 | Luxembourg | Den Atelier | Club | 800 Luf | 1100 | 470 |

Record: 1 of 11

**Exprimez les requêtes suivantes en langage SQL:**

1. Affichez toutes les informations pour les concerts qui ne sont pas à Luxembourg et dont le nombre de places est au moins 1000.

2. Affichez l'artiste, la date et la localité des concerts qui se tiendront à Luxembourg pendant la deuxième moitié de l'année.

3. Insérez le concert suivant:

| No. | Artiste | Date | Début | Localité | Lieu | Type Lieu | Places |
|---|---|---|---|---|---|---|---|
| 112 | Oasis | 12/09/1998 | 20:00 | Weiswampach | Fête sous tente | Autre | 1000 |

4. Affichez l'artiste, la date, la localité et le prix des concerts qui ont lieu dans un hall sportif ou un club à partir du 1/8/1998. Triez cette liste par ordre ascendant sur les types des lieux et à l'intérieur d'un type par ordre descendant sur le prix.

5. Quel était le prix moyen pour un concert en mois de juillet 1998 ? On ignorera les fêtes sous tente.

6. Classez les localités par ordre descendant sur le montant des recettes des concerts pour l'année 1998 (Recette d'un concert=Tickets vendus*Prix d'un ticket). Ignorez les localités pour lesquelles il n'y a pas encore de recettes.

7. Comptez le nombre de localités dans lesquelles a eu lieu un concert pendant les mois de juillet et août 1998.

8. Effacez tous les concerts qui ont eu lieu avant le 1/8/1998.

9. Affichez le nom de l'artiste, la date, la localité ainsi que le nombre de places encore disponibles pour les concerts qui auront lieu au mois de juillet 1998.

10. Un client achète 2 tickets pour le prochain concert de la "Kelly Family". Affichez d'abord une liste avec tous les concerts prévus pour cet artiste, et modifiez ensuite la table de façon à ce qu'elle reflète la vente des 2 tickets pour le concert correspondant.

    Nous supposons que cette requête donne comme résultat un seul concert avec le numéro **103** comme valeur de la clé primaire. Vous allez utiliser ce numéro pour identifier de façon unique et précise l'enregistrement qui doit refléter la vente des 2 tickets.

    Quel problème constatez-vous en ce qui concerne les valeurs des champs *fldPlaces* et *fldTicketsVendus* pour l'enregistrement 103 (voir exemples d'enregistrements) ?

    _____

    _____

11. Affichez la liste des localités, à l'exception de Luxembourg, dans lesquelles ont eu lieu au moins 2 concerts pendant la première moitié de l'année 1998. Indiquez pour chaque localité le nombre de concerts.

12. Le concert numéro 108 (voir exemples d'enregistrements) aura lieu au club "Den Atelier" à Luxembourg à la date et l'heure prévue initialement. Les tickets déjà vendus gardent leur validité et le prix d'un nouveau ticket ne change pas. Effectuez les modifications correspondantes dans la table.

## 7.3 Les requêtes SQL multitable

La plupart des BD réelles ne sont pas constituées d'une seule table, mais d'un ensemble de tables liées entre elles via certains champs (voir Chapitre 6.4). Par conséquent, les requêtes correspondantes ne sont pas ciblées sur une, mais sur plusieurs tables.

Nous allons différencier 2 méthodes pour lier plusieurs tables dans une requête:

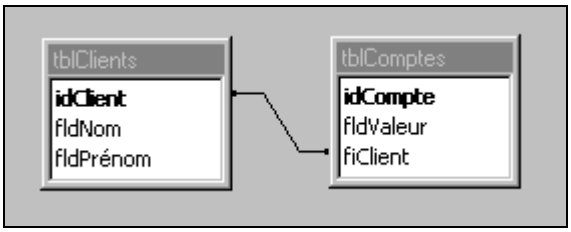
1. La **jointure**, qui lie plusieurs tables via des champs communs;

2. Les **requêtes imbriquées**, qui utilisent le résultat d'une requête comme source d'une autre.

## 7.3.1 La jointure

### *7.3.1.1 Exemple d'introduction*

Voici deux tables qui représentent une gestion (très simplifiée) des comptes d'une banque:




**tblComptes**

| idCompte | fldValeur | fiClient |
|---|---|---|
| 101 | 20000 | 3 |
| 106 | 48000 | 2 |
| 112 | 9000 | 3 |
| 125 | 5000 | 1 |

**tblClients**

| idClient | fldNom | fldPrénom |
|---|---|---|
| 1 | Pegaso | Emilio |
| 2 | Weber | Jos |
| 3 | Muller | Ketty |

 En principe, la présence d'une relation (clé étrangère/clé primaire) entre deux tables est une condition nécessaire pour effectuer une jointure sur les tables.

Une question possible serait:

- Affichez pour tous les comptes, le numéro de compte, la valeur actuelle, ainsi que le nom du client correspondant.

Voici la requête nécessaire pour répondre à notre question

```
SELECT tblcomptes.idCompte, tblComptes.fldValeur, tblClients.fldNom
FROM tblComptes, tblClients
WHERE tblComptes.fiClient=tblClients.idClient;
```

La clause **FROM** contient les deux tables impliquées dans la jointure.

La clause **WHERE** contient ce qu'on appelle la condition de jointure. Dans notre exemple, la condition de jointure demande l'égalité des valeurs pour les champs *fiClient* et *idClient*.

La clause **SELECT** contient les noms des champs à afficher.

Voici le résultat correspondant

| idCompte | fldValeur | fldNom |
|---|---|---|
| 101 | 20000 | Muller |
| 106 | 48000 | Weber |
| 112 | 9000 | Muller |
| 125 | 5000 | Pegaso |

Cette requête représente donc une jointure entre les table *tblComptes* et *tblClients*. Remarquez pour l'instant que nous avons préfixé chaque nom d'un champ par le nom de la table correspondante. Au moment où une requête porte sur plusieurs tables, on doit soit s'assurer que le nom de chaque champ est unique pour l'ensemble des tables, soit adopter la notation <Nom de la table>.<Nom du champ>. Puisque les noms des champs impliqués dans notre exemple sont tous différents, nous pouvons donc faciliter l'écriture de la requête:

```
SELECT idCompte, fldValeur, fldNom
FROM tblComptes, tblClients
WHERE fiClient=idClient;
```

Afin de comprendre le fonctionnement d'une jointure, et surtout celui de la condition de jointure, il est intéressant d'examiner en détail comment SQL procède à l'exécution d'une jointure. Pour cela, nous allons nous baser sur l'exemple précédent.

SQL exécute la requête en plusieurs étapes:

1. Comme la clause FROM contient 2 tables, SQL crée d'abord le **produit cartésien** des deux tables. Pour le produit cartésien, SQL associe à chaque enregistrement de la première table, tous les enregistrements de la deuxième table. Les enregistrements du produit cartésien contiennent donc les champs de la première table suivis des champs de la deuxième table. Si la première table contient N enregistrements et la deuxième table M enregistrements, alors le produit cartésien des deux tables est composé de N*M enregistrements.

Produit cartésien des tables *tblComptes* et *tblClients*:

| idCompte | fldValeur | fiClient | idClient | fldNom | fldPrénom |
|---|---|---|---|---|---|
| 101 | 20000 | 3 | 1 | Pegaso | Emilio |
| 101 | 20000 | 3 | 2 | Weber | Jos |
| 101 | 20000 | 3 | 3 | Muller | Ketty |
| 106 | 48000 | 2 | 1 | Pegaso | Emilio |
| 106 | 48000 | 2 | 2 | Weber | Jos |
| 106 | 48000 | 2 | 3 | Muller | Ketty |
| 112 | 9000 | 3 | 1 | Pegaso | Emilio |
| 112 | 9000 | 3 | 2 | Weber | Jos |
| 112 | 9000 | 3 | 3 | Muller | Ketty |
| 125 | 5000 | 1 | 1 | Pegaso | Emilio |
| 125 | 5000 | 1 | 2 | Weber | Jos |
| 125 | 5000 | 1 | 3 | Muller | Ketty |

**tblComptes** **tblClients**

2. Le produit cartésien contient un bon nombre d'enregistrements, qui ne donnent pas de sens logique.

| idCompte | fldValeur | fiClient | idClient | fldNom | fldprénom |
|---|---|---|---|---|---|
| 101 | 20000 | 3 | 1 | Pegaso | Emilio |
| 101 | 20000 | 3 | 2 | Weber | Jos |
| 101 | 20000 | **3** | **3** | Muller | Ketty |
| 106 | 48000 | 2 | 1 | Pegaso | Emilio |
| 106 | 48000 | **2** | **2** | Weber | Jos |
| 106 | 48000 | 2 | 3 | Muller | Ketty |
| 112 | 9000 | 3 | 1 | Pegaso | Emilio |
| 112 | 9000 | 3 | 2 | Weber | Jos |
| 112 | 9000 | **3** | **3** | Muller | Ketty |
| 125 | 5000 | **1** | **1** | Pegaso | Emilio |
| 125 | 5000 | 1 | 2 | Weber | Jos |
| 125 | 5000 | 1 | 3 | Muller | Ketty |

En fait, pour tous les enregistrements non marqués, *fiClient* ne correspond pas à *idClient*. Pour ces enregistrements, on associe donc un compte à un client qui n'est pas le propriétaire de ce compte.

C'est ici qu'intervient la condition de jointure, qui élimine du produit cartésien les enregistrements ne donnant pas de sens logique dans le contexte de la requête. Après avoir réalisé le produit cartésien, SQL **élimine tous les enregistrements qui ne correspondent pas à la condition de jointure de la clause WHERE**, donc tous les enregistrements pour lesquels l'expression logique **fiClient=idClient** n'est pas vraie.

Voici le résultat de l'application de la condition de jointure:

| idCompte | fldValeur | fiClient | idClient | fldNom | fldprénom |
|---|---|---|---|---|---|
| 101 | 20000 | 3 | 3 | Muller | Ketty |
| 106 | 48000 | 2 | 2 | Weber | Jos |
| 112 | 9000 | 3 | 3 | Muller | Ketty |
| 125 | 5000 | 1 | 1 | Pegaso | Emilio |

3. Finalement SQL affiche uniquement les champs indiqués dans la clause **SELECT**.

| idCompte | fldValeur | fldNom |
|---|---|---|
| 101 | 20000 | Muller |
| 106 | 48000 | Weber |
| 112 | 9000 | Muller |
| 125 | 5000 | Pegaso |

### *7.3.1.2 Création d'une jointure*

Pour créer une jointure, on doit:

- Indiquer dans la clause **FROM** les noms des tables impliquées
- Préciser dans la clause **WHERE** une condition de jointure
- Indiquer dans la clause **SELECT** les noms des champs à afficher

La **condition de jointure** spécifie généralement mais pas nécessairement une égalité entre une clé étrangère d'une table et la clé primaire d'une table correspondante.

**Remarque:**

La requête

```
SELECT tblcomptes.idCompte, tblComptes.fldValeur, tblClients.fldNom
FROM tblComptes, tblClients
WHERE tblComptes.fiClient=tblClients.idClient;
```

peut encore s'écrire d'une façon plus lisible en utilisant des **alias** pour les noms des tables.

Exemple:

```
SELECT Co.idCompte, Co.fldValeur, Cl.fldNom
FROM tblComptes Co, tblClients Cl
WHERE Co.fiClient=Cl.idClient;
```

Il suffit d'indiquer les alias derrière les noms des tables dans la partie FROM, afin de les utiliser dans l'ensemble de la requête. Lorsque vous définissez des alias dans la clause FROM, vous ne pouvez plus utiliser les noms des tables dans la requête.

Au cas où les noms des champs seraient tout à fait différents pour les deux tables, de façon à ce qu'il n'y ait aucune ambiguïté, on peut complètement laisser de côté les noms des tables resp. les alias.

Exemple:

```
SELECT idCompte, fldValeur, fldNom
FROM tblComptes, tblClients
WHERE fiClient=idClient;
```

Au cas où il existerait uniquement une ambiguïté pour un certain nombre de champs, il suffit de préfixer ceux-ci par un alias ou par le nom de la table correspondante.

**Remarque:**

La clause WHERE peut bien entendu définir des critères de sélection en combinaison avec la condition de jointure.

Exemple:

- Affichez pour les comptes ayant une valeur actuelle >= 10000, le numéro de compte, la valeur actuelle, ainsi que le nom du client correspondant.

```
SELECT idCompte, fldValeur, fldNom
FROM tblComptes, tblClients
WHERE fiClient=idClient AND fldValeur>=10000;
```

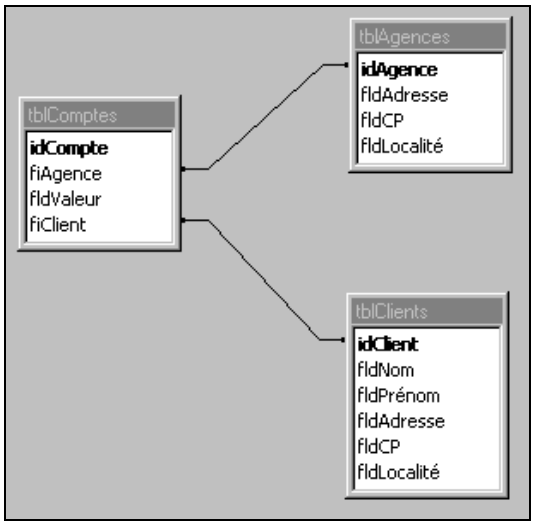
  Après avoir créé le produit cartésien, SQL élimine les enregistrements qui ne vérifient pas la condition de jointure (*fiClient=idClient*) et ceux qui ne vérifient pas le critère de sélection (*fldValeur*>=10000). Pour les enregistrements qui restent, SQL effectue l'affichage des champs demandés.

**Remarque:**

Une jointure peut mettre en relation plus que 2 tables.

Exemples:

Soient les 3 tables suivantes:




**tblComptes**

| idCompte | fiAgence | fldValeur | fiClient |
|---|---|---|---|
| 101 | 12 | 20000 | 3 |
| 106 | 24 | 48000 | 2 |
| 112 | 12 | 9000 | 3 |
| 125 | 30 | 5000 | 1 |

**tblClients**

| idClient | fldNom | fldPrénom | fldAdresse | fldCP | fldLocalité |
|---|---|---|---|---|---|
| 1 | Pegaso | Emilio | 25, rue de la Gare | 2278 | Diekirch |
| 2 | Weber | Jos | 66a, Cité Paerchen | 1234 | Schifflange |
| 3 | Muller | Ketty | 102, av G.Diederich | 6690 | Luxembourg |

**tblAgences**

| idAgence | fldAdresse | fldCP | fldLocalité |
|---|---|---|---|
| 12 | 15, bvd Royal | 5377 | Luxembourg |
| 24 | 67, rue de l'Alzette | 8765 | Esch-s-Alzette |
| 30 | 2, Grand Rue | 6678 | Ettelbruck |

1. Affichez pour tous les comptes, le numéro de compte, la valeur actuelle, le nom du client ainsi que sa localité et la localité de l'agence. Renommez les en-têtes de façon à ce qu'il n'y ait pas de confusion entre les données du client et celles de l'agence.

```
SELECT idCompte, fldValeur, fldNom, Cl.fldLocalité AS 'Localité Client',
   A.fldLocalité AS 'Localité Agence'
FROM tblAgences A, tblClients Cl, tblComptes
WHERE idAgence=fiAgence AND idClient=fiClient;
```

Comme la jointure contient 3 tables, nous avons 2 conditions de jointure.

 En général, on a:

**Nombre de conditions de jointure = Nombre de tables dans la jointure – 1**

2. Affichez le numéro de compte, le nom et prénom du client ainsi que le numéro d'agence pour les comptes dont l'agence se trouve dans la même localité ou habite le client correspondant.

```
SELECT idCompte, fldNom, fldPrénom, idAgence
FROM tblAgences A, tblClients Cl, tblComptes
WHERE idAgence=fiAgence AND idClient=fiClient AND
   A.fldLocalité=Cl.fldLocalité;
```

## 7.3.2 Auto- jointure

 Il est possible de définir **une jointure d'une table avec soi-même**. Dans ce cas, il faut obligatoirement utiliser des alias.

Exemple:

Soit la table

**tblComptes**

| idCompte | fiAgence | fldValeur | fiClient |
|---|---|---|---|
| 101 | 12 | 20000 | 3 |
| 106 | 24 | 48000 | 2 |
| 112 | 12 | 9000 | 3 |
| 125 | 30 | 5000 | 1 |

- Afficher le numéro de compte, et la valeur pour les comptes ayant une valeur supérieure à celle du compte 112

```
SELECT Co1.idCompte, Co1.fldValeur
FROM tblComptes Co1, tblComptes Co2
WHERE Co2.idCompte=112 AND Co1.fldValeur>Co2.fldValeur;
```

Cette requête ne semble pas évident à première vue. Nous allons analyser les étapes d'exécution de la requête.

1. Produit cartésien

| Co1.idCompte | Co1.fiAgence | Co1.fldValeur | Co1.fiClient | Co2.idCompte | Co2.fiAgence | Co2.fldValeur | Co2.fiClient |
|---|---|---|---|---|---|---|---|
| 101 | 12 | 20000 | 3 | 101 | 12 | 20000 | 3 |
| 106 | 24 | 48000 | 2 | 101 | 12 | 20000 | 3 |
| 112 | 12 | 9000 | 3 | 101 | 12 | 20000 | 3 |
| 125 | 30 | 5000 | 1 | 101 | 12 | 20000 | 3 |
| 101 | 12 | 20000 | 3 | 106 | 24 | 48000 | 2 |
| 106 | 24 | 48000 | 2 | 106 | 24 | 48000 | 2 |
| 112 | 12 | 9000 | 3 | 106 | 24 | 48000 | 2 |
| 125 | 30 | 5000 | 1 | 106 | 24 | 48000 | 2 |
| 101 | 12 | 20000 | 3 | 112 | 12 | 9000 | 3 |
| 106 | 24 | 48000 | 2 | 112 | 12 | 9000 | 3 |
| 112 | 12 | 9000 | 3 | 112 | 12 | 9000 | 3 |
| 125 | 30 | 5000 | 1 | 112 | 12 | 9000 | 3 |
| 101 | 12 | 20000 | 3 | 125 | 30 | 5000 | 1 |
| 106 | 24 | 48000 | 2 | 125 | 30 | 5000 | 1 |
| 112 | 12 | 9000 | 3 | 125 | 30 | 5000 | 1 |
| 125 | 30 | 5000 | 1 | 125 | 30 | 5000 | 1 |

Ce tableau associe tous les comptes 1 à 1.

2. Sélection des enregistrements

Il s'agit ici de la partie la plus délicate, puisque nous ne retrouvons plus une condition de jointure classique du type égalité - clé primaire/clé étrangère.

La sélection se fait en deux étapes. Comme nous voulons afficher tous les comptes ayant une valeur supérieure à celle du compte 112, nous allons uniquement garder les enregistrements pour lesquels un compte est associé au compte 112, c.à.d. les enregistrements pour lesquels le critère de sélection **Co2.idCompte=112** s'applique.

| Co1.idCompte | Co1.fiAgence | Co1.fldValeur | Co1.fiClient | Co2.idCompte | Co2.fiAgence | Co2.fldValeur | Co2.fiClient |
|---|---|---|---|---|---|---|---|
| 101 | 12 | 20000 | 3 | **101** | 12 | 20000 | 3 |
| 106 | 24 | 48000 | 2 | **101** | 12 | 20000 | 3 |
| 112 | 12 | 9000 | 3 | **101** | 12 | 20000 | 3 |
| 125 | 30 | 5000 | 1 | **101** | 12 | 20000 | 3 |
| 101 | 12 | 20000 | 3 | **106** | 24 | 48000 | 2 |
| 106 | 24 | 48000 | 2 | **106** | 24 | 48000 | 2 |
| 112 | 12 | 9000 | 3 | **106** | 24 | 48000 | 2 |
| 125 | 30 | 5000 | 1 | **106** | 24 | 48000 | 2 |
| 101 | 12 | 20000 | 3 | **112** | 12 | 9000 | 3 |
| 106 | 24 | 48000 | 2 | **112** | 12 | 9000 | 3 |
| 112 | 12 | 9000 | 3 | **112** | 12 | 9000 | 3 |
| 125 | 30 | 5000 | 1 | **112** | 12 | 9000 | 3 |
| 101 | 12 | 20000 | 3 | **125** | 30 | 5000 | 1 |
| 106 | 24 | 48000 | 2 | **125** | 30 | 5000 | 1 |
| 112 | 12 | 9000 | 3 | **125** | 30 | 5000 | 1 |
| 125 | 30 | 5000 | 1 | **125** | 30 | 5000 | 1 |

| Co1.idCompte | Co1.fiAgence | Co1.fldValeur | Co1.fiClient | Co2.idCompte | Co2.fiAgence | Co2.fldValeur | Co2.fiClient |
|---|---|---|---|---|---|---|---|
| 101 | 12 | 20000 | 3 | 112 | 12 | 9000 | 3 |
| 106 | 24 | 48000 | 2 | 112 | 12 | 9000 | 3 |
| 112 | 12 | 9000 | 3 | 112 | 12 | 9000 | 3 |
| 125 | 30 | 5000 | 1 | 112 | 12 | 9000 | 3 |

Ce tableau associe donc chaque compte (inclus le compte 112 même) au compte 112.

Il suffit maintenant de sélectionner les comptes qui ont une valeur supérieure à celle du compte 112. Ceci est fait à l'aide de la condition de jointure **Co1.fldValeur>Co2.fldValeur** Pour cet exemple, la condition de jointure ne se définit donc pas sur la clé étrangère/clé primaire.

| Co1.idCompte | Co1.fiAgence | Co1.fldValeur | Co1.fiClient | Co2.idCompte | Co2.fiAgence | Co2.fldValeur | Co2.fiClient |
|---|---|---|---|---|---|---|---|
| 101 | 12 | **20000** | 3 | 112 | 12 | **9000** | 3 |
| 106 | 24 | **48000** | 2 | 112 | 12 | **9000** | 3 |
| 112 | 12 | **9000** | 3 | 112 | 12 | **9000** | 3 |
| 125 | 30 | **5000** | 1 | 112 | 12 | **9000** | 3 |

| Co1.idCompte | Co1.fiAgence | Co1.fldValeur | Co1.fiClient | Co2.idCompte | Co2.fiAgence | Co2.fldValeur | Co2.fiClient |
|---|---|---|---|---|---|---|---|
| 101 | 12 | 20000 | 3 | 112 | 12 | 9000 | 3 |
| 106 | 24 | 48000 | 2 | 112 | 12 | 9000 | 3 |

3. Affichage des champs spécifiés

La dernière étape consiste dans l'affichage des champs indiqués dans la clause SELECT

| Co1.idCompte | Co1.fldValeur |
|---|---|
| 101 | 20000 |
| 106 | 48000 |

Avec l'auto-jointure, nous avons étudié un cas qui nous a montré que nous n'avons pas toujours une condition de jointure classique avec une égalité entre clé étrangère et clé primaire d'une table associée.

Une condition de jointure ne doit pas nécessairement impliquer une clé étrangère/clé primaire
Bien qu'une condition de jointure soit généralement définie à l'aide de l'opérateur d'égalité (=) , elle peut également être spécifiée à l'aide des opérateurs suivants:

- <>
- <
- >
- <=
- >=
- BETWEEN ... AND
- IN
- LIKE

Dans ce cas, on parle d'une jointure par non égalité. Ces conditions de jointure sont surtout employées en relation avec une auto-jointure.

Voici un autre exemple d'une auto-jointure:

- Affichez les numéros des comptes ayant une agence différente que le compte numéro 101.

```
SELECT CO1.idCompte
FROM tblComptes CO1, tblComptes CO2
WHERE CO2.idCompte=101 AND CO1.fiAgence<>CO2.fiAgence;
```

**A faire : Exercice 4**

## 7.3.3 Les requêtes imbriquées

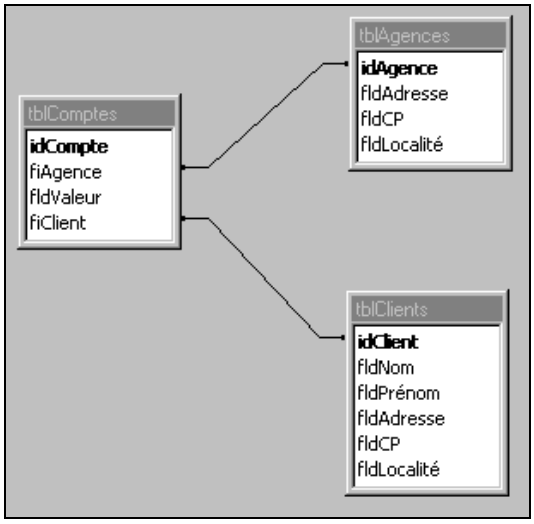
Nous savons qu'une requête de sélection se base sur une ou plusieurs tables pour afficher un résultat. En SQL, on peut imbriquer plusieurs requêtes, c.à.d. le résultat d'une requête imbriquée sert comme base pour une deuxième requête. Une requête imbriquée est encore parfois appelée 'SELECT interne' ou 'sous-requête'.

On distingue généralement deux types de requêtes imbriquées:
1. les requêtes imbriquées, qui renvoient comme résultat une seule valeur;
2. les requêtes imbriquées, qui renvoient comme résultat un ensemble de valeurs.

### *7.3.3.1 La requête imbriquée renvoie une seule valeur*

Exemple:

Soient les trois tables suivantes.




**tblComptes**

| idCompte | fiAgence | fldValeur | fiClient |
|---|---|---|---|
| 101 | 12 | 20000 | 3 |
| 106 | 24 | 48000 | 2 |
| 112 | 12 | 9000 | 3 |
| 125 | 30 | 5000 | 1 |

**tblClients**

| idClient | fldNom | fldPrénom | fldAdresse | fldCP | fldLocalité |
|---|---|---|---|---|---|
| 1 | Pegaso | Emilio | 25, rue de la Gare | 2278 | Diekirch |
| 2 | Weber | Jos | 66a, Cité Paerchen | 1234 | Schifflange |
| 3 | Muller | Ketty | 102, av G.Diederich | 6690 | Luxembourg |

**tblAgences**

| idAgence | fldAdresse | fldCP | fldLocalité |
|---|---|---|---|
| 12 | 15, bvd Royal | 5377 | Luxembourg |
| 24 | 67, rue de l'Alzette | 8765 | Esch-s-Alzette |
| 30 | 2, Grand Rue | 6678 | Ettelbruck |

La requête:

```
SELECT fldNom, fldPrénom
FROM tblClients
WHERE idClient = (SELECT fiClient
                  FROM tblComptes
                  WHERE idCompte=106);
```

retourne le nom et prénom du client qui est le propriétaire du compte numéro 106.

Analysons le fonctionnement de cette requête:

Le requête imbriquée:

```
SELECT fiClient
FROM tblComptes
WHERE idCompte=106;
```

retourne simplement la valeur 2

On peut donc traduire la requête de niveau supérieur en

```
SELECT fldNom, fldPrénom
FROM tblClients
WHERE idClient = 2;
```

Cette requête retourne finalement le nom et prénom du client correspondant, c.à.d. 'Weber' 'Jos'

La requête imbriquée doit **renvoyer au maximum une seule valeur**. Si tel n'est pas le cas, SQL ne pourra pas exécuter la requête de niveau supérieur, et génère un message d'erreur.

Dans la clause WHERE de la requête de niveau supérieur, le résultat de la requête imbriquée doit obligatoirement être comparé à un champ de type de données compatible avec la valeur retournée.

Comme la requête imbriquée doit retourner une seule valeur, on utilise souvent des fonctions d'agrégation dans la clause SELECT de la requête imbriquée.

On peut avoir plusieurs niveaux d'imbrication de requêtes. Une requête imbriquée peut donc déjà se baser sur le résultat d'une autre requête imbriquée

**Exemples:**

Reprenons les 3 tables *tblComptes*, *tblClients* et *tblAgences*

1. Affichez le numéro du(des) compte(s) avec la plus grande valeur.

```
SELECT idCompte
FROM tblComptes
WHERE fldValeur=(SELECT MAX(fldValeur)
                 FROM tblComptes);
```

Remarque:

Dans une requête imbriquée, vous n'avez pas besoin d'utiliser des alias lorsque la même table est utilisée plusieurs fois.

2. Affichez les numéros des comptes et la valeur actuelle pour les comptes dont la valeur est supérieure à la moyenne des valeurs.

```
SELECT idCompte, fldValeur
FROM tblComptes
WHERE fldValeur > (SELECT AVG(fldValeur)
                   FROM tblComptes);
```

3. Affichez le nom, le prénom, l'adresse, le code postal et la localité du client, qui possède le compte avec la plus petite valeur. Nous supposons qu'il existe uniquement un seul compte avec la plus petite valeur.

```
SELECT fldNom, fldPrénom, fldAdresse, fldCP, fldLocalité
FROM tblClients
WHERE idClient=(SELECT fiClient
                FROM tblComptes
                WHERE fldValeur=(SELECT MIN(fldValeur)
                                 FROM tblComptes));
```

4. Pour effectuer des statistiques, on vous demande la requête suivante. Affichez le numéro de compte et la valeur actuelle pour les comptes dont la valeur est plus petite que la moyenne des valeurs pour les comptes dont les clients habitent au Luxembourg, mais plus grande que la moyenne des valeurs pour les comptes dont les clients habitent à Diekirch ou Ettelbruck.

```
SELECT idCompte, fldValeur
FROM tblComptes
WHERE fldValeur< (SELECT AVG(fldValeur)
                  FROM tblComptes, tblClients
                  WHERE fiClient=idClient
                    AND fldLocalité='Luxembourg')
  AND fldValeur> (SELECT AVG(fldValeur)
                  FROM tblComptes, tblClients
                  WHERE fiClient=idClient AND fldLocalité IN
                     ('Diekirch','Ettelbruck'));
```

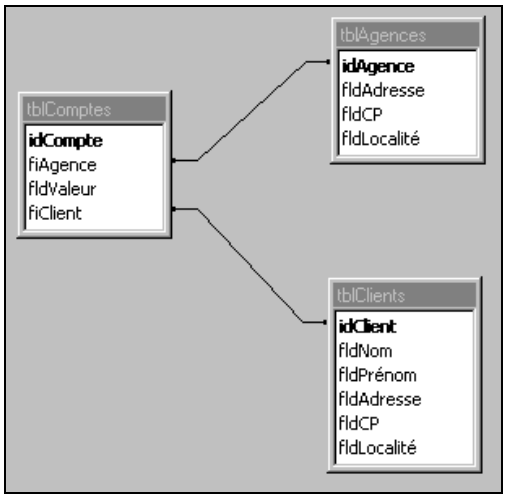
Remarque:

Comme cet exemple nous le montre, on peut avoir plusieurs requêtes imbriquées dans une seule clause WHERE.

### *7.3.3.2 La requête imbriquée renvoie un ensemble de valeurs*

Exemple:

Reprenons les trois tables suivantes




**tblComptes**

| idCompte | fiAgence | fldValeur | fiClient |
|---|---|---|---|
| 101 | 12 | 20000 | 3 |
| 106 | 24 | 48000 | 2 |
| 112 | 12 | 9000 | 3 |
| 125 | 30 | 5000 | 1 |

**tblClients**

| idClient | fldNom | fldPrénom | fldAdresse | fldCP | fldLocalité |
|---|---|---|---|---|---|
| 1 | Pegaso | Emilio | 25, rue de la Gare | 2278 | Diekirch |
| 2 | Weber | Jos | 66a, Cité Paerchen | 1234 | Schifflange |
| 3 | Muller | Ketty | 102, av G.Diederich | 6690 | Luxembourg |

**tblAgences**

| idAgence | fldAdresse | fldCP | fldLocalité |
|---|---|---|---|
| 12 | 15, bvd Royal | 5377 | Luxembourg |
| 24 | 67, rue de l'Alzette | 8765 | Esch-s-Alzette |
| 30 | 2, Grand Rue | 6678 | Ettelbruck |

La requête

```
SELECT idCompte, fldValeur
FROM tblComptes
WHERE fiClient IN (SELECT idClient
                   FROM tblClients
                   WHERE fldLocalité='Luxembourg' OR
                     fldLocalité='Diekirch');
```

retourne le numéro de compte et la valeur actuelle pour les comptes dont le client habite à Luxembourg ou Diekirch

Analysons le fonctionnement de cette requête:

Le requête imbriquée:

```
SELECT idClient
FROM tblClients
WHERE fldLocalité='Luxembourg' OR fldLocalité='Diekirch';
```

retourne tous les numéros de clients habitant à Luxembourg ou Diekirch. Cette requête retourne donc l'ensemble de valeurs [1, 3].

On peut donc traduire la requête de niveau supérieur en

```
SELECT idCompte, fldValeur
FROM tblComptes
WHERE fiClient IN (1, 3);
```

Cette requête retourne finalement le numéro de compte et la valeur des comptes correspondants.

| idCompte | fldValeur |
|---|---|
| 101 | 20000 |
| 112 | 9000 |
| 125 | 5000 |

La requête imbriquée renvoie un ensemble de n valeurs. Cet ensemble peut bien sûr être vide (n=0) ou être composé d'une seule valeur (n=1).

Dans la clause WHERE de la requête de niveau supérieur, le champ pour lequel on vérifie l'appartenance à l'ensemble de valeurs retourné par la sous-requête, doit avoir un type de données compatible avec les valeurs de l'ensemble.

Parfois, il est convenable d'utiliser l'option DISTINCT dans la clause SELECT de la sous-requête, afin d'éviter des doublons dans l'ensemble résultat. Toutefois, ceci est uniquement une mesure d'optimisation des requêtes imbriquées.

On peut avoir plusieurs niveaux d'imbrication de requêtes. Une requête imbriquée peut donc déjà se baser sur le résultat d'une autre requête imbriquée

Exemples:

Reprenons les 3 tables *tblComptes*, *tblClients* et *tblAgences*

1. Affichez les numéros des comptes qui sont gérés par une agence située à Luxembourg.

```
SELECT idCompte
FROM tblComptes
WHERE fiAgence IN (SELECT idAgence
                   FROM tblAgences
                   WHERE fldLocalité='Luxembourg');
```

2. Affichez le nom et le prénom de tous les clients ayant un compte géré par une agence située à Luxembourg ou à Esch-s-Alzette.

```
SELECT fldNom, fldPrénom
FROM tblClients
WHERE idClient IN (SELECT fiClient
                   FROM tblComptes
                   WHERE fiAgence IN (SELECT idAgence
                                      FROM tblAgences
                                      WHERE fldLocalité IN ('Luxembourg',
                                                            'Esch-s-Alzette')));
```

3. Affichez le nom et le prénom de tous les clients n'ayant pas de compte.

```
SELECT fldNom, fldPrénom
FROM tblClients
WHERE idClient NOT IN (SELECT DISTINCT fiClient
                       FROM tblComptes);
```

Remarque:

A l'intérieur d'une requête imbriquée, on peut faire référence à un champ d'une table définie dans la requête de niveau supérieur. Dans ce cas on parle d'une **requête imbriquée corrélée**. Une valeur retournée par ce type de requête dépend donc d'un champ qui reçoit ses valeurs à partir d'une requête de niveau supérieur.

Exemple:

Affichez le nom et le prénom des clients ayant au moins un compte avec une valeur de 9000 Luf.

```
SELECT fldNom, fldPrénom
FROM tblClients C
WHERE 9000 IN (SELECT fldValeur
               FROM tblComptes
               WHERE fiClient=C.idClient);
```

L'ensemble de valeurs retourné du SELECT imbriqué dépend de la valeur de *C.idClient*. SQL exécute cette requête de la façon suivante:

- *C.idClient* est substitué par sa valeur pour le premier enregistrement de la table *tblClients*. *C.idClient* prend donc la valeur 1.
- La requête imbriquée

```
SELECT fldValeur
FROM tblComptes
WHERE fiClient=1;
```

  est exécutée avec comme résultat l'ensemble [5000].
- La requête de niveau supérieur ne retourne donc aucun résultat

- *C.idClient* est maintenant substitué par sa valeur pour le deuxième enregistrement de la table *tblClients***.** *C.idClient* prend donc la valeur 2**.**
- La requête imbriquée

```
SELECT fldValeur
FROM tblComptes
WHERE fiClient=2;
```

  retourne l'ensemble [48000]
- La requête de niveau supérieur ne retourne donc aucun résultat

- *C.idClient* est ensuite substitué par sa valeur pour le troisième enregistrement de la table *tblClients***.** *C.idClient* prend donc la valeur 3.
- La requête imbriquée

```
SELECT fldValeur
FROM tblComptes
WHERE fiClient=3;
```

- retourne l'ensemble [20000 , 9000]

- La requête de niveau supérieur retourne le résultat 'Muller' 'Ketty' puisque effectivement le troisième enregistrement de la table *tblClients* contient une valeur de *idClient* (3) , qui produit dans la requête imbriquée un ensemble contenant la valeur 9000.


Formulez cette requête sans utiliser le principe de la requête corrélée


**A faire : Exercice 5**

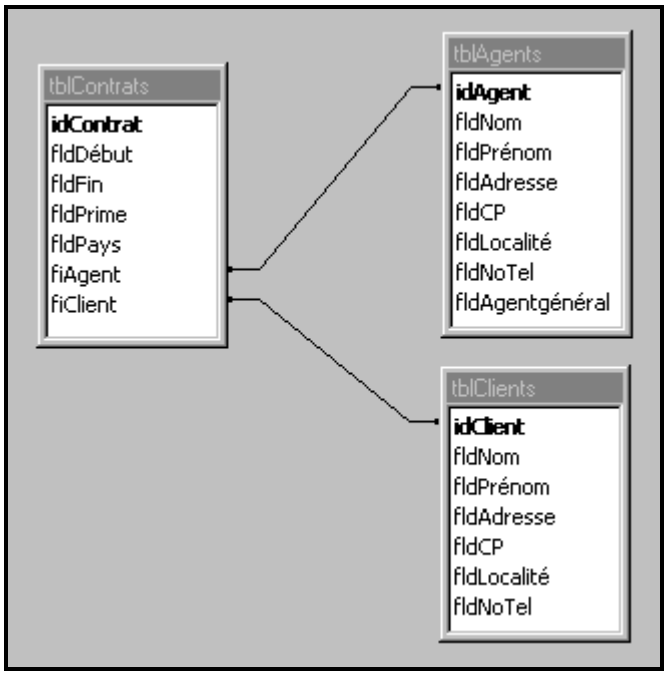

**Répétition générale en SQL : Exercice 6 et Exercice 7**

## 7.3.4 Exercices SQL

### Exercice 4: Assurance bagages (Les jointures)

Une société d'assurances offre une formule 'Assurance Bagages'. Cette formule garantit pendant une durée limitée un remboursement intégral de la valeur des bagages avec contenu en cas de vol ou de perte.

Voici la BD utilisée pour gérer ce type d'assurances.




**Remarques:**

- Comme certains noms de champs sont identiques pour les tables *tblAgents* et *tblClients*, vous devez veiller à employer les noms des tables resp. des alias aux bons endroits dans les requêtes.
- Le champ *fldAgentgénéral* est du type booléen (valeurs VRAI/FAUX resp. YES/NO)

**Utilisez le mécanisme des jointures afin de répondre aux questions suivantes.**

1. Affichez pour les contrats qui couvrent la France comme pays de destination, le nom de l'agent.

2. Affichez le numéro de contrat, les dates de début et de fin du contrat ainsi que le nom, prénom, adresse, code postal et localité du client pour tous les contrats qui couvrent la période entre le 14 juillet et le 20 juillet 1998 et dont le pays de destination était l'Italie. Utilisez des alias partout dans la requête.

3. Déterminez la plus grande prime parmi celles où le pays de destination est la Belgique et l'agent n'est pas un agent général.

4. Affichez le numéro de contrat, la prime, le nom et prénom du client ainsi que le nom et prénom de l'agent pour tous les contrats ou l'agent a le même nom que le client.

5. Affichez toutes les informations concernant les clients ayant un agent qui habite à Capellen. Eliminez un effet indésirable qui peut se produire à cause du fait qu'un client peut avoir conclu plusieurs contrats avec le même agent.

6. Affichez pour chaque client, le numéro de client, son nom, le nom de son agent et la somme des primes de tous les contrats qu'il a conclu avec cet agent. Appliquez une clause GROUP BY sur le résultat de la jointure des tables impliquées. Au cas ou un client a conclu des contrats avec plusieurs agents différents, vous devez afficher un groupe pour chaque agent.

7. Soient les valeurs suivantes pour les deux tables *tblContrats* et *tblAgents*:

| idContrat | fldDébut | fldFin | fldPrime | fldPays | fiAgent | fiClient |
|---|---|---|---|---|---|---|
| 1000 | 22/07/1998 | 03/08/1998 | 2300 | France | 2 | 10 |
| 1001 | 12/07/1998 | 22/07/1998 | 1750 | Italie | 1 | 11 |
| 1002 | 19/07/1998 | 18/08/1998 | 4500 | Belgique | 1 | 11 |

| idAgent | fldNom | fldPrénom | fldAdresse | fldCP | fldLocalité | fldNoTel | fldAgentGénéral |
|---|---|---|---|---|---|---|---|
| 1 | Pezzotto | Alfredo | 23, rue de Mamer | 6555 | Capellen | 238987 | No |
| 2 | Kremer | Pierre | 2, bvd de la liberté | 9980 | Luxembourg | 228890 | Yes |

Expliquez pour la requête suivante, les étapes d'exécution, en précisant à chaque fois les résultats intermédiaires.

```
SELECT IdContrat, fldPrime, fldNom
FROM tblContrats, tblAgents
WHERE fiAgent=idAgent
 AND fldAgentGénéral=No;
```

8. Elaborez une liste qui affiche pour chaque agent le nombre de contrats par pays de destination.

9. Indiquez le nom, le prénom, l'adresse, le code postal et la localité des clients ayant conclu un contrat qui a une prime strictement inférieure à celle du contrat numéro 1003.

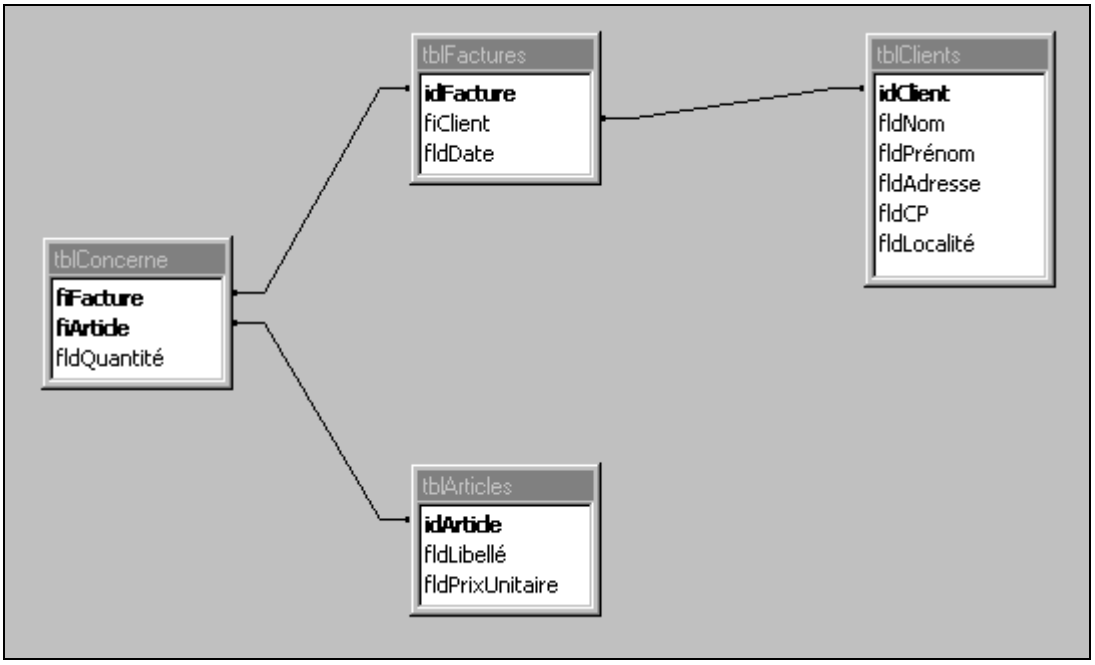
10. Classez les agents par ordre descendant sur le nombre de contrats qu'ils ont conclu. En tenant uniquement compte des agents qui ont conclu au moins 2 contrats, affichez pour chaque agent, son numéro, son nom et prénom ainsi que le nombre de contrats qu'il a conclu.

11. Affichez le nom et prénom des agents ayant conclu un contrat avec un client, qui a encore conclu un contrat avec au moins un autre agent.

12. Affichez le nom et prénom de tous les agents ayant conclu un contrat avec un client habitant dans la même localité que le client numéro 11.

# Exercice 5: Facturation (Les requêtes imbriquées)

Un magasin spécialisé dans la vente d'appareils électroménagers entretient la BD suivante afin de gérer la facturation.




**Utilisez le mécanisme des requêtes imbriquées afin de répondre aux questions suivantes.**

1. Affichez le libellé et le prix unitaire de l'article (des articles) qui est le plus cher.

2. Affichez le numéro de l'article ainsi que le libellé pour les articles moins cher que le prix moyen de tous les articles.

3. Affichez le numéro et la date de toutes les factures dont le client habite à Luxembourg.

4. Affichez le nom et le prénom des clients qui habitent à Luxembourg et qui sont concernés par une facture etablie au cours du mois d'août 1998.

5. Affichez le numéro et le libellé des articles qui sont plus cher que le prix moyen de tous les articles, et pour lesquels il existe une ou plusieurs factures avec une quantité >1.

6. Affichez le nom, le prénom, l'adresse, le code postal et la localité de tous les clients ayant déjà acheté un article plus cher que 3000 Luf.

   Exprimez la même requête sans utiliser les requêtes imbriquées

7. Affichez le nom, le prénom, l'adresse, le code postal et la localité de tous les clients ayant uniquement acheté des articles plus cher que 3000 Luf.

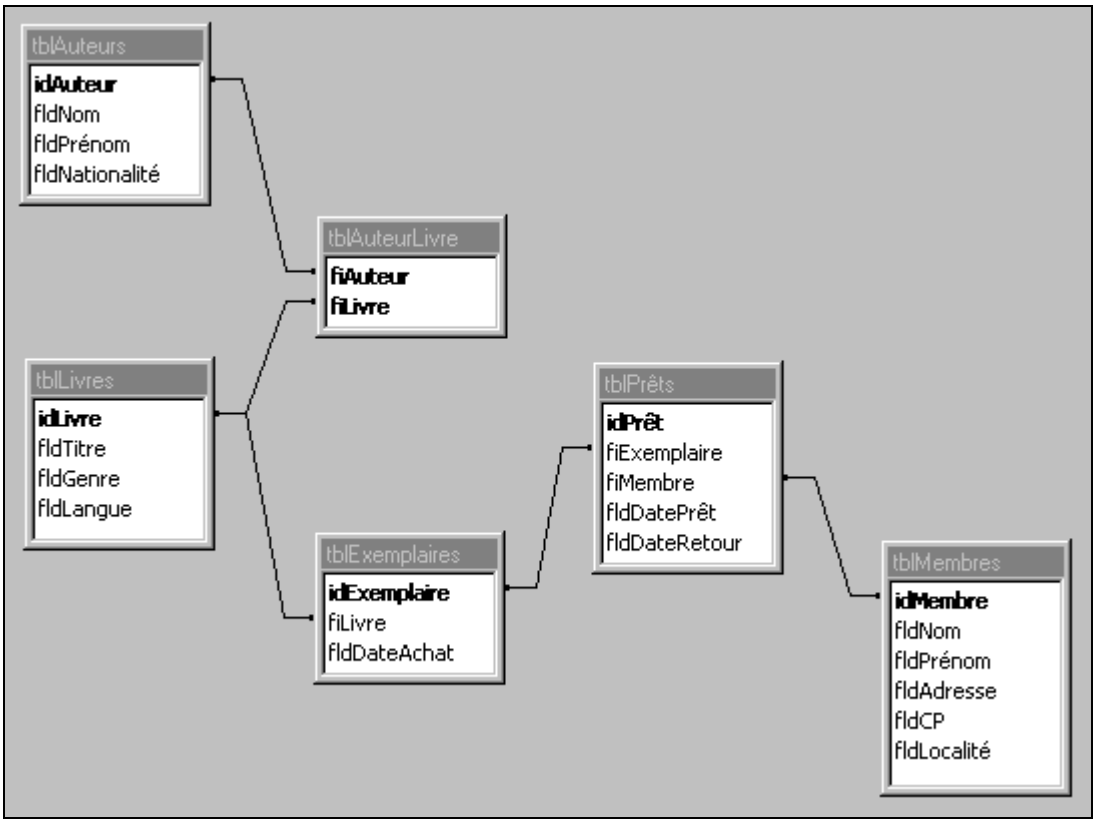
Proposez une solution alternative en vous servant du mécanisme de la requête imbriquée corrélée.

8. Affichez le nom, le prénom, l'adresse, le code postal et la localité de tous les clients ayant déjà acheté pour une somme > 3000 Luf. par facture. Utilisez au maximum possible les requêtes imbriquées.

9. Affichez le nom et le prénom de tous les clients ayant une facture, qui concerne un seul article. La facture ne doit donc ni concerner plusieurs articles différents ni avoir une quantité >1 pour un seul article.

# Exercice 6: Bibliothèque

Une bibliothèque utilise la BD suivante.




**Remarques:**

- Un auteur peut rédiger plusieurs livres et un livre peut être rédigé par plusieurs auteurs.
- La bibliothèque peut disposer de plusieurs exemplaires du même livre.
- Un prêt concerne un seul exemplaire d'un livre.
- Le champ *fldDateRetour* de la table *tblPrêts* reste indéterminé (NULL) tant que l'exemplaire emprunté n'a pas été retourné à la bibliothèque.

**Formulez en SQL les requêtes suivantes:**

1. Affichez le numéro, le titre et le genre de tous les livres allemands (Code=ALL). Classez la liste par ordre alphabétique sur le genre (p.ex. 'Roman' avant 'Technique') et à l'intérieur d'un genre par ordre ascendant sur les numéros.

2. Affichez une liste triée par ordre alphabétique de tous les genres de livres disponibles.

3. Affichez une liste de toutes les localités où habite un membre dont l'adresse contient l'abréviation 'bvd' , indiquant que le membre habite sur un boulevard.

4. Affichez toutes les informations de la table *tblAuteurs* concernant les auteurs ayant une des nationalités suivantes.

| Nationalité (Pays d'origine) | Code |
|---|---|
| **Allemagne** | **ALL** |
| **Angleterre** | **ANG** |
| **France** | **FRA** |
| **Autriche** | **AUT** |
| **Italie** | **ITA** |
| **Suisse** | **SUI** |
| **Russie** | **RUS** |

5. Effacez tous les membres n'ayant pas encore effectué un prêt.

6. Affichez le nom, le prénom, l'adresse, le code postal et la localité de tous les membres habitant à Luxembourg ou à Esch-s-Alzette, n'ayant pas encore retourné un exemplaire emprunté.

7. Affichez pour chaque livre le titre, le genre, la langue et le nombre d'exemplaires disponibles (emprunté ou non). Triez la liste par ordre alphabétique sur la langue, sur le genre et finalement sur le titre. Le champ indiquant le nombre d'exemplaires disponibles doit avoir l'en-tête 'Exemplaires disponibles'.

8. Affichez le nom et le prénom des auteurs ayant écrit un livre français dont le titre contient le mot 'passage', et dont la bibliothèque possède au moins 3 exemplaires.

9. Affichez tous les livres (Titre et genre) de l'auteur 'Alexandre Dumas'. Triez la liste par ordre alphabétique sur le titre.

10. Affichez le nom, le prénom et le nombre de prêts effectués, pour tous les membres qui habitent à Esch-s-Alzette ou à Luxembourg, ayant déjà effectué au moins 2 prêts. Triez la liste par ordre alphabétique sur le nom.

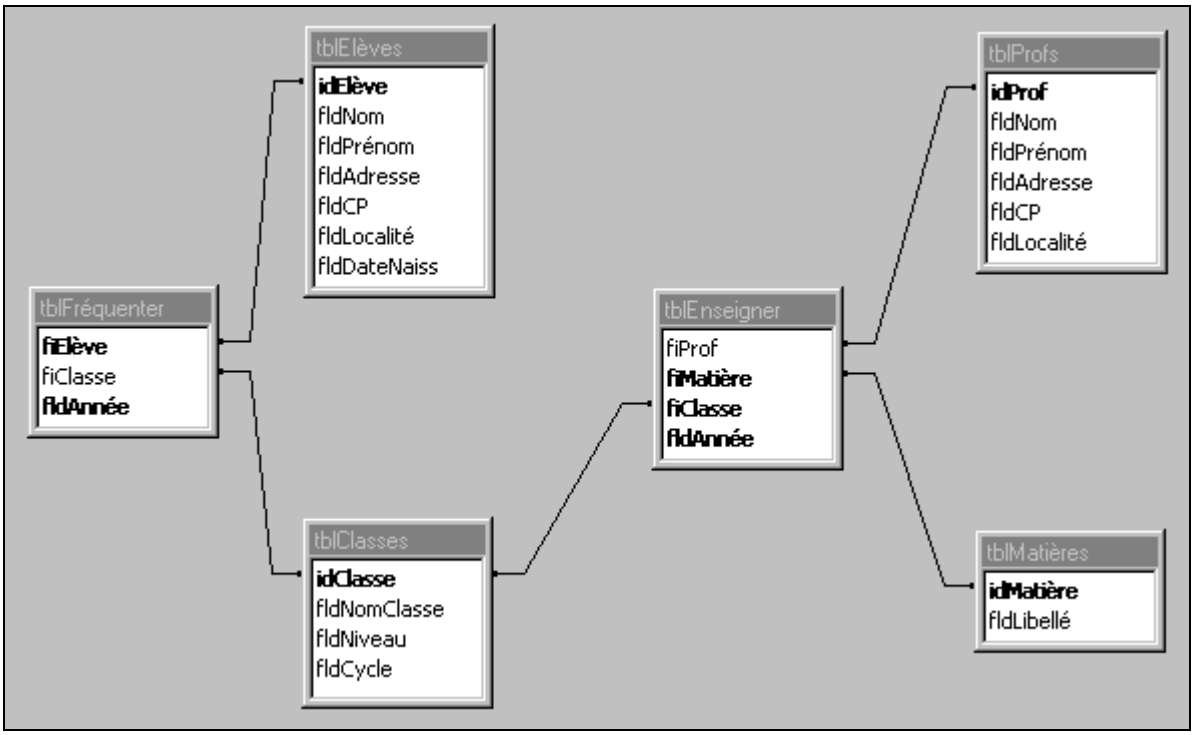
11. Créez une liste qui affiche pour chaque exemplaire actuellement emprunté (pas encore retourné), le numéro du prêt, le numéro, le nom et le prénom du membre ayant emprunté le livre ainsi que le titre et le genre du livre en question. Triez la liste par ordre alphabétique sur le nom et le prénom du membre.

12. Quels sont les auteurs (Nom et prénom) ayant déjà écrit un livre ensemble avec l'auteur 'Margaret Gibson' ?

13. Quels sont les auteurs (Nom et prénom) n'ayant pas encore écrit un livre ensemble avec l'auteur 'Margaret Gibson' ?

## Exercice 7: Gestion d'une école

Voici une BD qui représente une gestion simplifiée des cours d'un lycée technique.




**Remarques:**

- Une classe est représentée par un code interne (*idClasse*) , un nom de classe (*fldNomClasse*) tel que '13CG2' ou '11CM1' , un niveau (*fldNiveau*) tel que 10 pour la classe '10GE2' ou 13 pour '13CG1' , et un champ indiquant le cycle (*fldCycle*) avec les valeurs possibles 'Inférieur', 'Moyen' et 'Supérieur'.
- Nous supposons qu'un élève ne change pas de classe pendant l'année scolaire. Les champs *fiElève* et *fldAnnée* forment donc la clé primaire de la table *tblFréquenter*. Cependant, un élève peut fréquenter la même classe pendant plusieurs années consécutives (redoublants).
- De même nous supposons qu'une matière est enseignée pendant une année par un seul prof dans une classe. Les champs *fiMatière*, *fiClasse* et *fldAnnée* forment donc la clé primaire de la table *tblEnseigner*. Toutefois, un prof peut enseigner la même matière pendant plusieurs années dans une même classe ou la même matière pendant une année dans plusieurs classes.
- Les champs *fldAnnée* des tables *tblfréquenter* et *tblEnseigner* font référence à des années scolaires. On y retrouve des valeurs telles que '97/98' ou '95/96'. La BD ne contient pas uniquement la situation de l'année scolaire actuelle, mais également celle des années précédentes.

**Formulez en SQL les requêtes suivantes:**

1. Affichez pour l'année scolaire '97/98' , le nom de chaque classe ainsi que le nombre d'élèves.

2. Affichez par année scolaire et par niveau le nombre d'élèves. Triez la liste par ordre ascendant sur l'année scolaire et par ordre ascendant sur le niveau.

3. Affichez le nom et le prénom de tous les profs ayant enseigné une matière dans une classe de 13ème pendant les 5 dernières années scolaires (à partir de l'année scolaire '97/98'). Triez la liste par ordre alphabétique sur le nom du prof.

4. Dressez une liste avec le nom, le prénom, l'adresse, le code postal, et la localité pour tous les élèves qui ont fréquenté la classe '08TH1' pendant l'année scolaire '96/97'. La liste doit être triée par ordre alphabétique sur le nom des élèves. Utilisez au maximum possible le mécanisme des requêtes imbriquées.

5. Créez une liste, qui montre pour l'année scolaire '97/98', pour chaque classe, les matières enseignées avec les noms et prénoms des profs correspondants. Triez la liste par ordre alphabétique sur les noms des classes et à l'intérieur d'une classe par ordre alphabétique sur les matières. Utilisez uniquement des jointures en définissant des alias pour toutes les tables impliquées.

6. Créez une liste des profs (nom & prénom) qui est triée par ordre descendant sur le nombre de cours enseignés pendant les 3 dernières années scolaires (à partir de l'année scolaire '97/98'). La notion de cours est définie par le fait d'enseigner une matière dans une classe.

7. Affichez le nom et le prénom des profs qui enseignent au moins une matière dans une classe pendant l'année scolaire '97/98'.

   Formulez la même requête en utilisant le mécanisme de la requête imbriquée corrélée.

8. Affichez le nom, le prénom, l'adresse, le code postal et la localité de tous les élèves ayant fréquenté pendant l'année scolaire 96/97 une classe du cycle inférieur. Utilisez au maximum les requêtes imbriquées.

9. Affichez le nom, le prénom et la dénomination de la classe actuelle des élèves qui sont actuellement (Année '97/98') des redoublants. Attention: Un élève est actuellement un redoublant s'il a fréquenté l'année scolaire passée une classe de même niveau, mais pas nécessairement la même classe.

10. Sachant qu'une classe ne devrait avoir un effectif supérieur à 21 élèves, le directeur vous demande d'établir une liste avec les noms des classes du cycle inférieur, qui pourraient encore accepter des nouveaux élèves pendant l'année scolaire '97/98'. Utilisez uniquement des requêtes imbriquées.

    Formulez la même requête en utilisant uniquement des jointures

    Formulez la même requête en utilisant le mécanisme de la requête corrélée.

11. Affichez le nom et le prénom, ainsi que le nom, le niveau et le cycle de leur classe actuelle (année = '97/98') de tous les élèves qui n'ont jamais redoublé une classe dans notre lycée.

12. Affichez parmi tous les profs, qui ont déjà enseigné la même matière que le prof numéro 10001, ceux n'ayant pas encore enseigné la même matière au même niveau que le prof numéro 10001 pendant les années scolaires '96/97' et '97/98'.

# 7.4 La méthode QBE

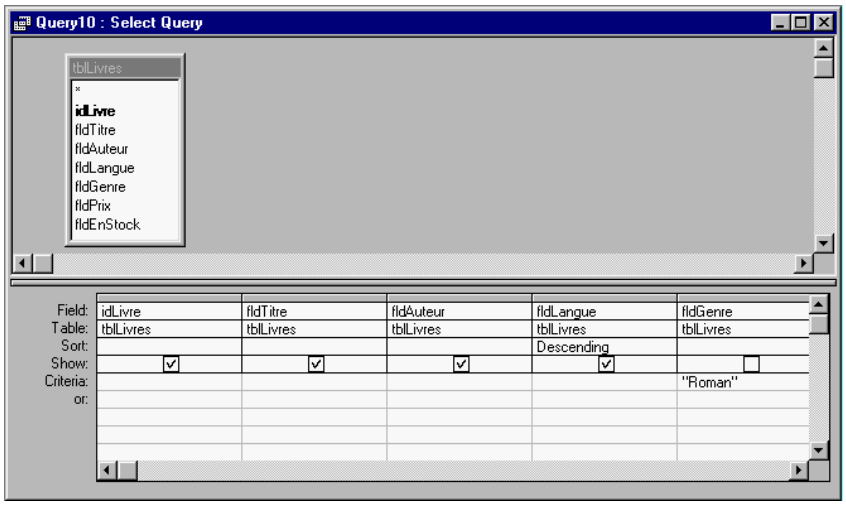
Bien que le langage SQL soit le standard unanime en ce qui concerne l'extraction de données d'une BD ainsi que leur manipulation, les informaticiens étaient déjà pendant les années '70 à la recherche d'une possibilité de créer des requêtes sans faire recours à un langage d'interrogation. Etant d'accord sur la flexibilité et les nombreuses possibilités de SQL, on voulait quand même combler au grand désavantage de ce langage, à savoir une syntaxe assez rigide et surtout pas uniforme à travers les différents SGBD.

Les chercheurs voulaient créer une possibilité de spécifier graphiquement tous les éléments d'une requête c.à.d. la ou les tables cibles, les critères de sélection et les champs concernés. Le standard **QBE (Query By Example)** était né. Pourtant, QBE tout comme SQL n'est pas implémenté de façon uniforme dans les différents SGBD. Ce n'est qu'en 1985, que QBE devenait vraiment populaire avec son introduction dans le SGBD PARADOX, qui fut commercialisé par la société BORLAND. Actuellement, tous les SGBD qui tournent sous une interface graphique du type Windows offrent le système QBE. Citons par exemple dBASE, Visual FOXPRO, Superbase et surtout MS-Access qui offre actuellement selon les experts l'implémentation la plus conviviale du standard QBE.

Prenons comme exemple les requêtes de sélection. QBE offre à l'utilisateur une interface graphique qui lui permet de :

- Sélectionner une table sur laquelle la requête sera basée (→SQL : FROM …).
- Choisir parmi les champs de cette table ceux qui vont être affiché (→ SQL : SELECT …).
- Définir pour un ou plusieurs champs des critères de sélection (→ SQL : WHERE …).
- Définir un ordre de tri (→ SQL : ORDER BY …).
- etc.

Voici à titre d'exemple un écran QBE de MS Access :




La requête correspondante en SQL serait:

```
SELECT idLivre, fldTitre, fldAuteur, fldLangue
FROM tblLivres
WHERE fldGenre="Roman"
```

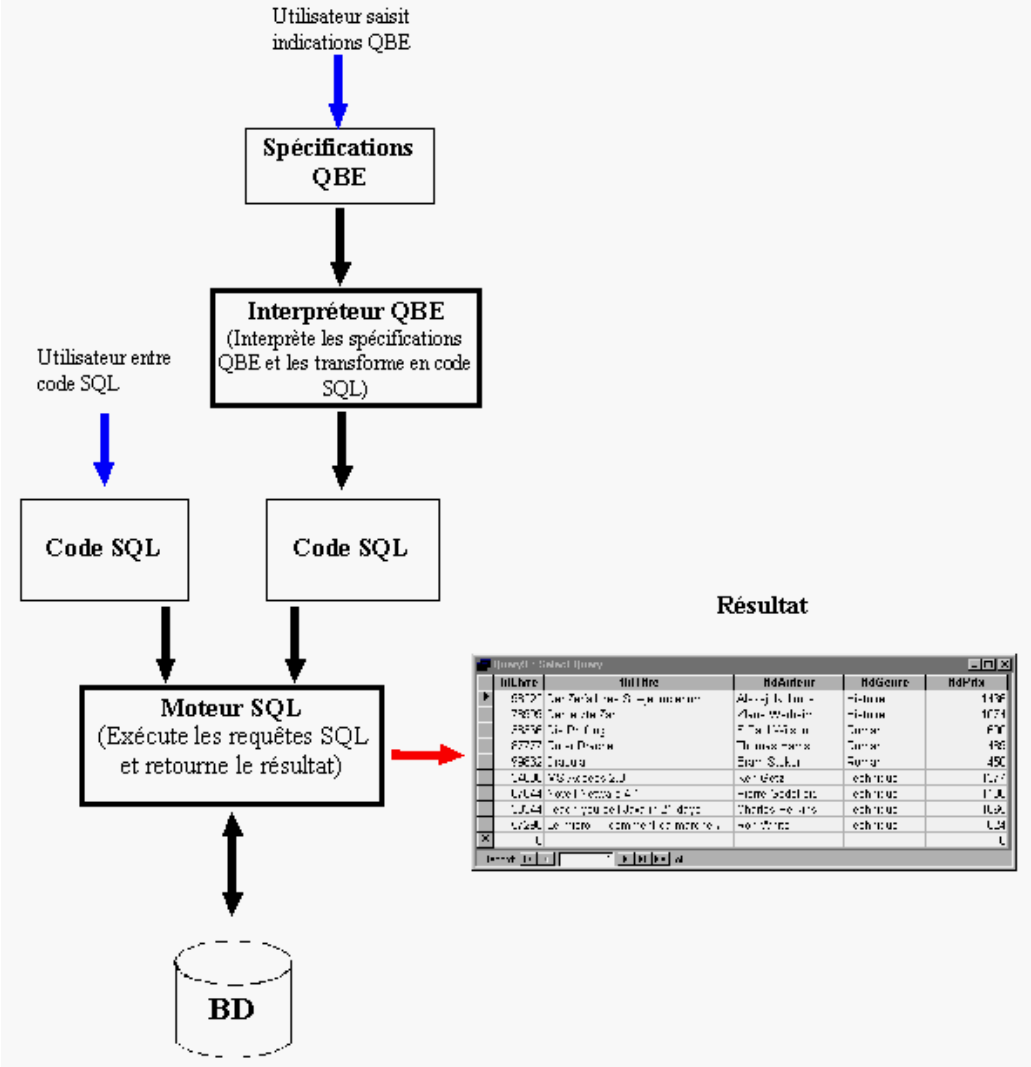
```
ORDER BY fldLangue DESC;
```

Les SGBD actuels offrent de plus en plus des possibilités QBE avancées telles que l'utilisation des fonctions d'agrégation, l'implémentation des requêtes d'insertion, de modification et de suppression etc. .

Référez-vous à la documentation de votre SGBD pour voir comment QBE est implémenté et quelles sont les fonctionnalités et les limites.

Il est cependant important de savoir que les requêtes QBE sont toujours exécutées via SQL, parce qu'un SGBD ne comprend pas vraiment QBE. QBE n'est qu'une interface graphique couplée à un interpréteur, qui transforme les indications de l'écran QBE en SQL. La partie du SGBD, qui exécute la requête (appelée le moteur SQL), utilise le code SQL généré par l'interpréteur de la même façon que celui entré directement par l'utilisateur.

Nous avons l'architecture suivante:




## A faire : TP No 3 & TP No4

# 8. Les formulaires (angl. forms)

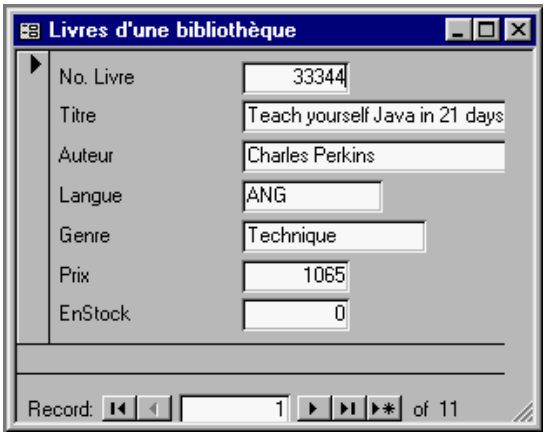
## 8.1 Définition

L'affichage, la saisie et la modification des données ont été réalisés jusqu'à maintenant directement dans les tables. Même les requêtes ont soit affiché leurs résultats sous forme de feuilles de données ( = sous-table), soit modifié directement le contenu des tables. Les formulaires représentent un autre outil de manipulation de données des SGBD.

Un formulaire est une aide utile pour consulter et modifier rapidement et facilement les données d'une table. Les diverses facilités mises à notre disposition par les formulaires nous offrent un bon confort ainsi qu'une très grande sécurité des données lors des manipulations.

**En général on utilise un formulaire pour:**
- **Entrer des données.**
- **Consulter des données.**
- **Modifier des données.**

Voici à titre d'exemple un formulaire, qui affiche toutes les données d'une table qui contient des livres:

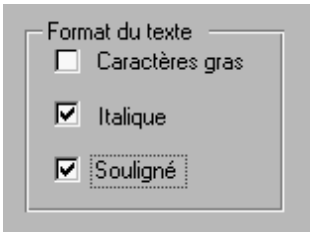
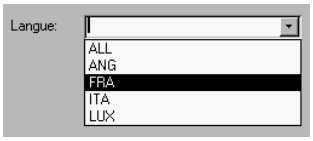



Vous remarquez que ce formulaire affiche **un** enregistrement à la fois.

Un formulaire est toujours lié à une table ou bien à une requête. Il ne représente donc qu'une interface entre l'utilisateur et les tables. Toutes les données saisies sur un formulaire sont donc inscrites dans la (les) table(s) correspondante(s).

Chaque formulaire est composé de contrôles. Voici une liste non exhaustive des contrôles les plus répandus dans les SGBD actuels:

| Nom du contrôle | Description | Utilisation |
|---|---|---|
| **Etiquette**<br>(angl. Label)<br><br>Exemple:<br><br>  | Affiche du texte fixe. | Ce type de contrôle n'est pas lié à un champ d'une BD. Il sert uniquement à fournir des informations à l'utilisateur. |
| **Zone de texte**<br>(angl. Text Box)<br><br>Exemple:<br><br>  | Contient des données de la BD. Ce contrôle affiche par exemple la valeur d'un champ pour l'enregistrement actuel. | Peut représenter des champs de tout type. |
| **Bouton d'options**<br>(angl. Option Button ou Radio Button)<br><br>Exemple:<br><br>  | Utilisés en groupe, ces boutons permettent de choisir une seule valeur parmi plusieurs possibles. Un bouton sélectionné signifie que la valeur associée à ce bouton est sélectionnée comme valeur pour le champ correspondant au groupe de boutons.<br>Les options dans un groupe représentent donc les valeurs possibles pour UN champ donné de la table.<br><br>Exemple: Le bouton *Féminin* sélectionné veut dire que le sexe de cet employé est féminin. | Ce contrôle représente de préférence des champs de type numérique, texte ou date.<br>On utilise des groupes de boutons d'options pour représenter des champs pouvant contenir seulement quelques valeurs prédéfinies, qui ne changent pas souvent ou pas du tout comme par exemple le sexe (masculin/féminin), le résultat d'un examen (Admin/Ajourné/Ecarté) etc. . |

| | | |
|---|---|---|
| **Case à cocher**<br>(angl. Check Box)<br><br>Exemple:<br>  <br> | Utilisé pour afficher le contenu d'un champ de type Oui/Non (Yes/No). La différence par rapport aux boutons d'option est qu'il est possible de cocher simultanément plusieurs cases dans un groupe. En plus, les cases à cocher apparaissent souvent seules et indépendant d'un groupe.<br>Chaque case concerne UN champ de la table.<br><br>Exemple: La table contient 3 champs à valeurs Oui/Non (*fldCaracGras*, *fldItalique*, *fldSouligné*). | Représente des champs à valeurs logiques (Oui/Non). |
| **Zone de liste**<br>(angl. List Box)<br><br>Exemple:<br>  <br> | Permet d'afficher une liste de valeurs parmi lesquelles l'utilisateur peut en choisir une.<br>On utilise des zones de liste pour représenter des champs qui contiennent plusieurs valeurs possibles. Lorsque la nature des données fait que des nouvelles options deviennent indispensables, il suffit de les ajouter dans la liste et chaque utilisateur pourra les sélectionner. | Ce contrôle représente de préférence des champs de type numérique, texte ou date.<br>On utilise des zones de liste pour représenter des champs pouvant contenir beaucoup de valeurs qui ne changent pas souvent ou pas du tout comme par exemple les noms des différents pays de l'Europe. |
| **Liste modifiable**<br>(angl. Combo Box)<br><br>Exemple:<br>  <br> | Combinaison entre une zone de liste et une zone de texte. L'utilisateur peut sélectionner une valeur de la liste ou entrer un texte de son choix. | Ce contrôle représente de préférence des champs de type numérique, texte ou date.<br>Utilisation pareille à la zone de liste mais avec l'option pour l'utilisateur d'entrer une valeur non prédéfinie. |

| | | |
|---|---|---|
| **Bouton de commande** (angl. Command Button)<br><br>Exemples:<br><br>  | Exécuter une ou plusieurs commandes systèmes respectivement lancer des modules de programmes créés par l'utilisateur.<br>Exemple1: Visualiser toutes les commandes d'un client.<br>Exemple2: Arrêter l'action en cours. | Ce type de contrôle n'est pas lié à un champ d'une BD. |

La plupart des SGBD offrent encore des contrôles pour améliorer la présentation des formulaires (contrôles graphiques, images, liens OLE …).

**Convention des noms:**

Les noms des formulaires sont précédés du préfixe **frm** (angl.: Form)

**Quand est-ce qu'on utilise des formulaires ?**

- Lorsqu'on ne veut pas que les utilisateurs travaillent directement dans les tables. Les formulaires offrent généralement des mécanismes de sécurité plus sophistiqués tels que les zones de listes qui empêchent les utilisateurs d'entrer n'importe quelle valeur dans un champ etc.
- Lorsqu'on veut présenter les données sous une forme plus conviviale. On peut par exemple utiliser des cases à cocher pour les champs à valeur Oui/Non (Yes/No).
- Lorsqu'on désire afficher les enregistrements un à la fois (→ Formulaires Colonne Simple)

**Le principe de conception d'un bon formulaire est toujours de minimiser le nombre de frappes afin de minimiser les erreurs possibles.**

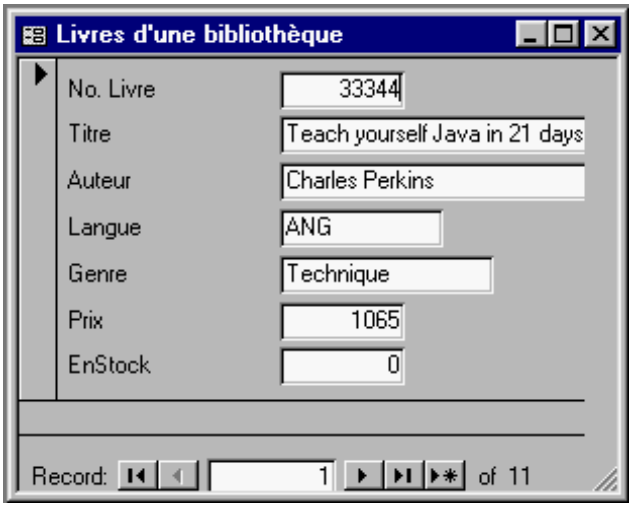
**Quelle est la base de création d'un formulaire ?**

- Tout comme les tables et les requêtes, un formulaire est un composant d'une BD, qui doit être crée et défini avant de pouvoir être utilisé pour manipuler les données.
- Chaque formulaire se crée à partir d'une table ou d'une requête.
- Les données affichées dans un formulaire proviennent donc de tables ou de requêtes, tandis que certaines informations spécifiques à l'apparence du formulaire (p.ex. couleur de l'arrière plan …) sont stockées dans la définition du formulaire.

## 8.2 Types de formulaires

En général nous distinguons 3 types de formulaires:

1. Formulaire Colonne Simple (angl. Single Column Form)

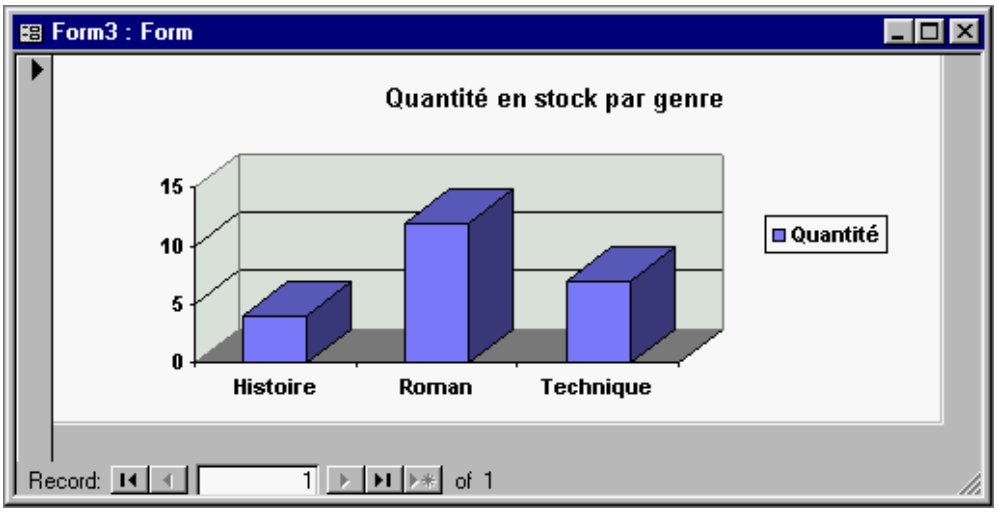



Dans un formulaire Colonne Simple, les valeurs des enregistrements sont affichées dans une seule colonne. Chaque valeur d'un enregistrement se trouve dans un champ de formulaire dédié. Un seul enregistrement est donc représenté à chaque fois.

2. Formulaire Tabulaire (angl. Tabular Form)


Dans un formulaire tabulaire, les enregistrements sont représentés sur des lignes et des colonnes. Ce type de formulaire a une apparence similaire à celle de la vue d'un tableau ou d'un résultat d'une requête.

3. Formulaire Graphique (angl. Graphical Form)




La plupart des SGBD offrent une multitude d'options de représentations graphiques (Histogrammes 2D, Histogrammes 3D, Diagrammes circulaires …).

**A faire : TP No 5**

# 8.3 Création d'un formulaire

Avant de créer un formulaire, quelques réflexions s'imposent:

- Comment est-ce qu'on veut représenter les données et quel type de formulaire est le plus adéquat ?
- Est-ce que l'utilisateur aura la possibilité d'ajouter, de modifier respectivement de supprimer des données ?
- Quels sont les contrôles appropriés pour représenter les différents champs de la table respectivement de la requête ?

 **Voici quelques règles générales d'utilisation des différents contrôles standard.**

- **Pour représenter un champ à valeur logique (Oui/Non), employez impérativement une case à cocher. Plusieurs cases à cocher peuvent être regroupées afin de représenter plusieurs champs à valeur logique.**

  Exemple: 
  
  L'utilisateur, qui est dans ce cas un employé d'une société d'assurances, peut indiquer si un client à inclus dans son contrat une assurance auto supplémentaire du type "Défense & Recours" .

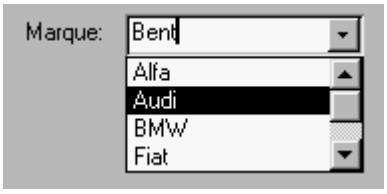
- **Pour représenter un champ, qui ne peut contenir qu'un nombre très limité (max 5) de valeurs prédéfinies du type numérique, texte ou date, qui sont en plus mutuellement exclusives, utilisez un groupe de boutons d'options.**

  Exemple: 
  
  L'employé choisit si la carte verte est envoyée à l'agent ou directement au client.

- **Un champ, qui peut contenir un nombre limité (> 5) de valeurs prédéfinies du type numérique, texte ou date, qui sont en plus mutuellement exclusives, devra être représenté par une zone de liste.**

  Exemple: 
  
  L'employé peut étendre la couverture de l'assurance auto sur un pays supplémentaire.

- **Lorsque pour un champ, représenté normalement par une zone de liste, vous voulez donner à l'utilisateur la possibilité d'entrer des valeurs outres que celles prédéfinies, utilisez une liste modifiable.**

Exemple: 


L'employé peut soit sélectionner une des marques prédéfinies, soit entrer lui-même un nom de marque.

- **Pour les champs ou vous ne pouvez pas du tout anticiper les valeurs, et qui ne sont pas du type logique, utilisez une zone de texte.**

Exemple: 


L'employé doit entrer le nom du client.

**Lors de la conception d'un formulaire, le respect de ces quelques règles garantit à l'utilisateur le principe de la saisie minimale. Partout ou une sélection de valeurs prédéfinies est possible, l'utilisateur n'a pas besoin d'entrer les données au clavier.**

**Avantages:**
1. **La rapidité de la saisie des données augmente → meilleure productivité.**
2. **Elimination de beaucoup de sources d'erreur.**

## A faire : TP No 6

## A faire : TP No 7

# 9. Les rapports (angl. reports)

## 9.1 Définition

Avec les formulaires, nous avons introduit un outil puissant pour consulter et manipuler les données d'une BD. Il est également possible d'imprimer les formulaires, mais les SGBD nous offrent un outil beaucoup plus puissant en termes de fonctionnalités pour imprimer les données et effectuer des calculs sur ces données. Il s'agit des rapports (ou états) (angl. reports), qui ont l'avantage d'être très flexibles en ce qui concerne la création de listes et de statistiques imprimées, mais qui ne permettent pas de dialogue interactif avec l'utilisateur. L'important pour l'utilisateur d'une BD est donc de savoir quand il faut utiliser un formulaire et quand un rapport.

**En général, on utilise un rapport pour:**

- **Imprimer des listes et statistiques concernant les données;**
- **Regrouper les données et créer des calculs sur les données;**
- **Créer des factures, bons de livraisons et autres pièces de gestion importantes.**

Reprenons notre table avec les livres d'une librairie

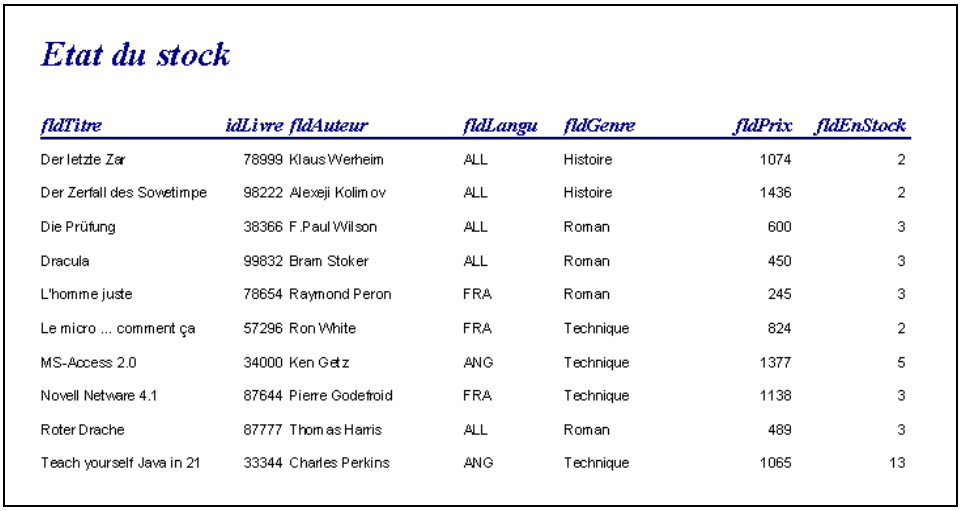
tblLivres : Table

| idLivre | fldTitre | fldAuteur | fldLangue | fldGenre | fldPrix | fldEnStock |
|---|---|---|---|---|---|---|
| 33344 | Teach yourself Java in 21 days | Charles Perkins | ANG | Technique | 1065 | 13 |
| 34000 | MS-Access 2.0 | Ken Getz | ANG | Technique | 1377 | 5 |
| 38366 | Die Prüfung | F.Paul Wilson | ALL | Roman | 600 | 3 |
| 57296 | Le micro ... comment ça marche ? | Ron White | FRA | Technique | 824 | 2 |
| 78654 | L'homme juste | Raymond Peron | FRA | Roman | 245 | 3 |
| 78999 | Der letzte Zar | Klaus Werheim | ALL | Histoire | 1074 | 2 |
| 87644 | Novell Netware 4.1 | Pierre Godefroid | FRA | Technique | 1138 | 3 |
| 87777 | Roter Drache | Thomas Harris | ALL | Roman | 489 | 3 |
| 98222 | Der Zerfall des Sowetimperiums | Alexeji Kolimov | ALL | Histoire | 1436 | 2 |
| 99832 | Dracula | Bram Stoker | ALL | Roman | 450 | 3 |
| 0 | | | | | 0 | 0 |

Record: 1 of 10

Exemple 1:

Le rapport suivant affiche simplement une liste avec tous les livres en stock. Cette liste est triée par ordre alphabétique sur le titre.

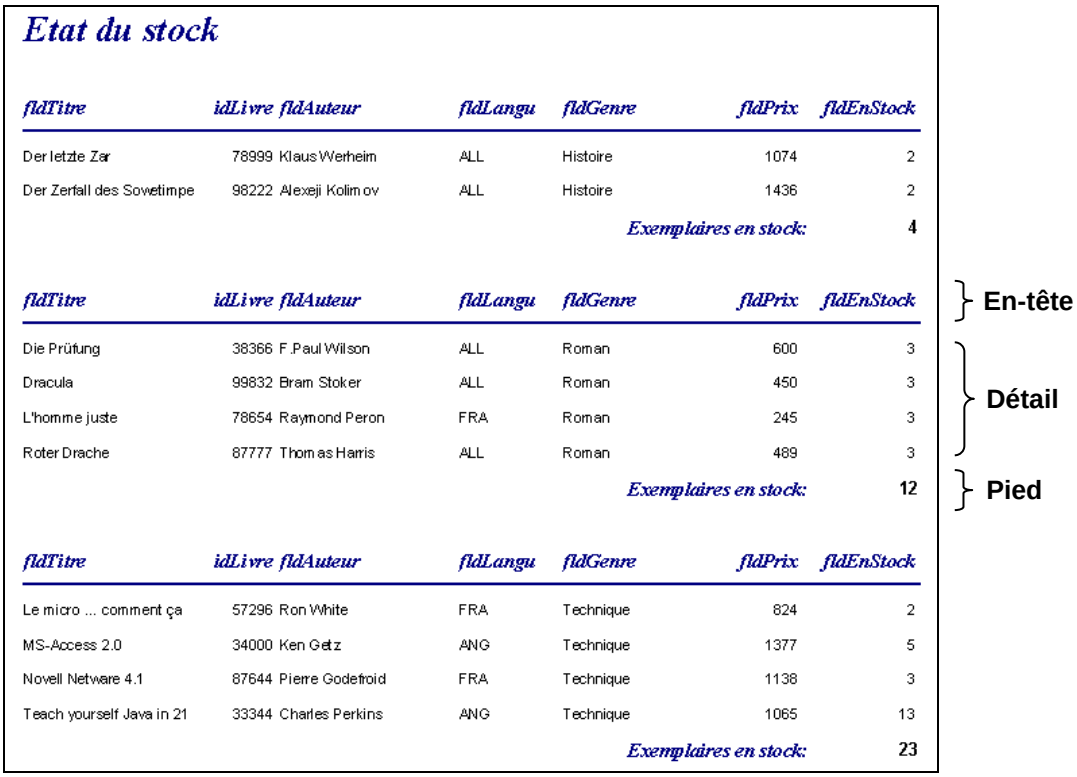


*Etat du stock*

| fldTitre | idLivre | fldAuteur | fldLangu | fldGenre | fldPrix | fldEnStock |
|---|---|---|---|---|---|---|
| Der letzte Zar | 78999 | Klaus Werheim | ALL | Histoire | 1074 | 2 |
| Der Zerfall des Sowetimpe | 98222 | Alexeji Kolimov | ALL | Histoire | 1436 | 2 |
| Die Prüfung | 38366 | F.Paul Wilson | ALL | Roman | 600 | 3 |
| Dracula | 99832 | Bram Stoker | ALL | Roman | 450 | 3 |
| L'homme juste | 78654 | Raymond Peron | FRA | Roman | 245 | 3 |
| Le micro ... comment ça | 57296 | Ron White | FRA | Technique | 824 | 2 |
| MS-Access 2.0 | 34000 | Ken Getz | ANG | Technique | 1377 | 5 |
| Novell Netware 4.1 | 87644 | Pierre Godefroid | FRA | Technique | 1138 | 3 |
| Roter Drache | 87777 | Thomas Harris | ALL | Roman | 489 | 3 |
| Teach yourself Java in 21 | 33344 | Charles Perkins | ANG | Technique | 1065 | 13 |

Exemple 2:

Un SGBD nous offre généralement la possibilité de regrouper les données. Chaque groupe est défini selon les valeurs d'un ou de plusieurs champs. Un groupe contient normalement 3 parties; une en-tête de groupe, une section détail et un pied de groupe. Dans notre exemple, nous allons créer des groupes basés sur la valeur du champ *fldGenre*, donc un groupe par genre. Pour chaque groupe, donc pour chaque genre, nous allons afficher les libellés des champs dans l'en-tête du groupe et les livres appartenant au groupe dans la section détail. A la fin de chaque groupe (dans le pied de groupe) sera affiché en plus, le total des exemplaires en stock pour ce groupe.



*Etat du stock*

| fldTitre | idLivre | fldAuteur | fldLangu | fldGenre | fldPrix | fldEnStock |
|---|---|---|---|---|---|---|
| Der letzte Zar | 78999 | Klaus Werheim | ALL | Histoire | 1074 | 2 |
| Der Zerfall des Sowetimpe | 98222 | Alexeji Kolimov | ALL | Histoire | 1436 | 2 |
| | | | | *Exemplaires en stock:* | | **4** |

| fldTitre | idLivre | fldAuteur | fldLangu | fldGenre | fldPrix | fldEnStock |
|---|---|---|---|---|---|---|
| Die Prüfung | 38366 | F.Paul Wilson | ALL | Roman | 600 | 3 |
| Dracula | 99832 | Bram Stoker | ALL | Roman | 450 | 3 |
| L'homme juste | 78654 | Raymond Peron | FRA | Roman | 245 | 3 |
| Roter Drache | 87777 | Thomas Harris | ALL | Roman | 489 | 3 |
| | | | | *Exemplaires en stock:* | | **12** |

| fldTitre | idLivre | fldAuteur | fldLangu | fldGenre | fldPrix | fldEnStock |
|---|---|---|---|---|---|---|
| Le micro ... comment ça | 57296 | Ron White | FRA | Technique | 824 | 2 |
| MS-Access 2.0 | 34000 | Ken Getz | ANG | Technique | 1377 | 5 |
| Novell Netware 4.1 | 87644 | Pierre Godefroid | FRA | Technique | 1138 | 3 |
| Teach yourself Java in 21 | 33344 | Charles Perkins | ANG | Technique | 1065 | 13 |
| | | | | *Exemplaires en stock:* | | **23** |

Exemple 3:

Dans ce rapport, les livres sont groupés par genre et à l'intérieur d'un genre par langue. Chaque groupe est donc défini par le genre **et** la langue.

**Etat du stock**

| fldTitre | idLivre | fldAuteur | fldLangue | fldGenre | fldPrix | fldEnStock |
|---|---|---|---|---|---|---|
| Der letzte Zar | 78999 | Klaus Werheim | ALL | Histoire | 1074 | 2 |
| Der Zerfall des Sowetimpe | 98222 | Alexeji Kolimov | ALL | Histoire | 1436 | 2 |
| | | | | *Exemplaires en stock:* | | 4 |

| fldTitre | idLivre | fldAuteur | fldLangue | fldGenre | fldPrix | fldEnStock |
|---|---|---|---|---|---|---|
| Die Prüfung | 38366 | F.Paul Wilson | ALL | Roman | 600 | 3 |
| Dracula | 99832 | Bram Stoker | ALL | Roman | 460 | 3 |
| Roter Drache | 87777 | Thomas Harris | ALL | Roman | 489 | 3 |
| | | | | *Exemplaires en stock:* | | 9 |

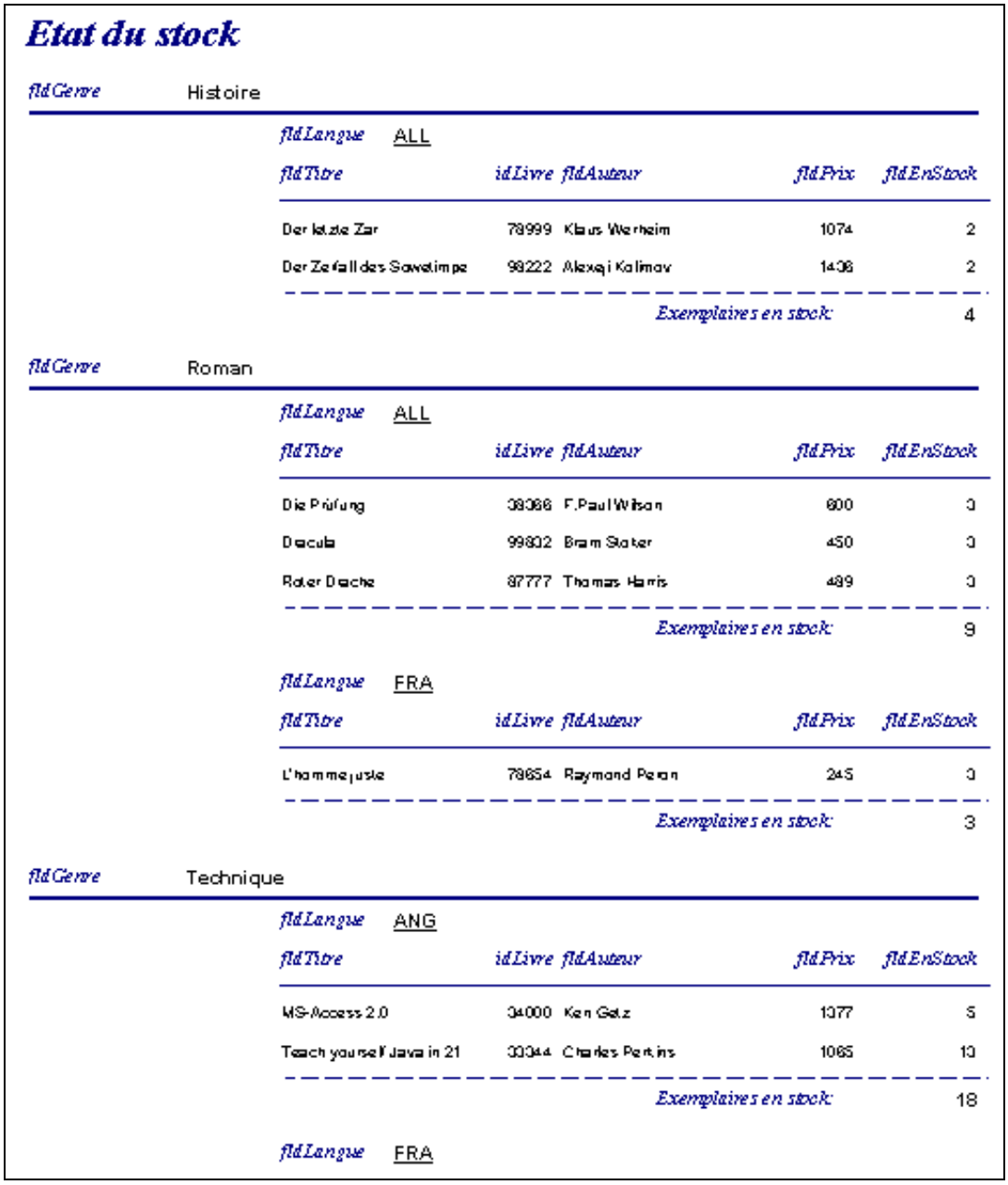
| fldTitre | idLivre | fldAuteur | fldLangue | fldGenre | fldPrix | fldEnStock |
|---|---|---|---|---|---|---|
| L'homme juste | 78654 | Raymond Peron | FRA | Roman | 245 | 3 |
| | | | | *Exemplaires en stock:* | | 3 |

| fldTitre | idLivre | fldAuteur | fldLangue | fldGenre | fldPrix | fldEnStock |
|---|---|---|---|---|---|---|
| MS-Access 2.0 | 34000 | Ken Getz | ANG | Technique | 1377 | 5 |
| Teach yourself Java in 21 | 33344 | Charles Perkins | ANG | Technique | 1065 | 13 |
| | | | | *Exemplaires en stock:* | | 18 |

| fldTitre | idLivre | fldAuteur | fldLangue | fldGenre | fldPrix | fldEnStock |
|---|---|---|---|---|---|---|
| Le micro ... comment ça | 57296 | Ron White | FRA | Technique | 824 | 2 |
| Novell Netware 4.1 | 87644 | Pierre Godefroid | FRA | Technique | 1138 | 3 |
| | | | | *Exemplaires en stock:* | | 5 |

Exemple 4:

On pourrait envisager de représenter le même groupement (genre & langue) d'une autre façon.



**Etat du stock**

fldGenre Histoire

fldLangue ALL

| fldTitre | idLivre | fldAuteur | fldPrix | fldEnStock |
|---|---|---|---|---|
| Der letzte Zar | 78999 | Klaus Werheim | 1074 | 2 |
| Der Zerfall des Sowetimpe | 98222 | Alexej Kolimov | 1436 | 2 |
| | | Exemplaires en stock: | | 4 |

fldGenre Roman

fldLangue ALL

| fldTitre | idLivre | fldAuteur | fldPrix | fldEnStock |
|---|---|---|---|---|
| Die Prüfung | 38088 | F.Paul Wilson | 600 | 3 |
| Dracula | 99802 | Bram Stoker | 450 | 3 |
| Roter Drache | 87777 | Thomas Harris | 489 | 3 |
| | | Exemplaires en stock: | | 9 |

fldLangue FRA

| fldTitre | idLivre | fldAuteur | fldPrix | fldEnStock |
|---|---|---|---|---|
| L'homme juste | 78654 | Raymond Peron | 245 | 3 |
| | | Exemplaires en stock: | | 3 |

fldGenre Technique

fldLangue ANG

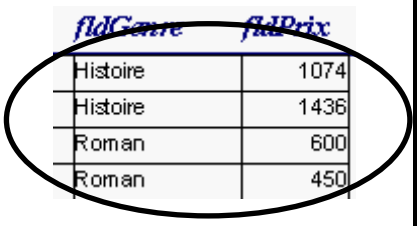
| fldTitre | idLivre | fldAuteur | fldPrix | fldEnStock |
|---|---|---|---|---|
| MS-Access 2.0 | 34000 | Ken Getz | 1377 | 5 |
| Teach yourself Java in 21 | 33044 | Charles Perkins | 1065 | 10 |
| | | Exemplaires en stock: | | 18 |

fldLangue FRA

**Quelle est la base de création d'un rapport ?**

- Tout comme les tables, les requêtes et les formulaires, un rapport est un composant d'une BD, qui doit être créé et défini avant de pouvoir être utilisé pour afficher les données et les calculs sur les données.
- Chaque rapport se crée à partir d'une table ou d'une requête.
- Les données affichées dans un rapport proviennent donc de tables ou de requêtes, tandis que certaines informations spécifiques à l'apparence du rapport (p.ex. Titre dans l'en-tête …) sont stockées dans la définition du rapport.

Chaque rapport est composé de contrôles. Puisque les rapports ne sont pas prévus pour le dialogue interactif avec l'utilisateur, ils contiennent dans la plupart des cas seulement 3 types de contrôles:

| Nom du contrôle | Description | Exemple |
|---|---|---|
| **Zone de texte** (angl. Text Box) | Affiche les données de la BD, ainsi que les résultats de calculs sur ces données. Les zones de textes constituent les contrôles les plus importants et les plus utilisés dans les rapports. |  |
| **Etiquette** (angl. Label) | Affiche du texte fixe. |  |
| **Contrôles graphiques** (angl. Graphical Controls) | Ces contrôles, dont le seul but est l'amélioration de la présentation des rapports peuvent être de types différents, comme par exemple des lignes , des éléments graphiques élémentaires tels que carrés ou rectangles, mais également des images importées. |  |

Néanmoins, beaucoup de SGBD prévoient également l'utilisation d'autres contrôles, comme par exemple les boutons d'options ou les cases à cocher.

**Convention des noms:**

Les noms des rapports sont précédés du préfixe **rpt** (angl.: report)

**Structure d'un rapport**

**Chaque rapport est subdivisé en différentes parties, appelés sections. Un rapport peut contenir les sections suivantes:**

- **En-tête/Pied de rapport**

L'en-tête de rapport apparaît une seule fois au début de la première page, et le pied de rapport apparaît une seule fois à la fin de la dernière page. L'en-tête de rapport est souvent utilisé pour afficher des logos ou la date actuelle. Le pied de rapport contient souvent des grand totaux.

- **En-tête/Pied de page**

Contient du texte, qui sera affiché/imprimé à chaque nouvelle page du rapport. L'en-tête de page contient généralement les noms des champs affichés dans la section détail. Le pied de page est souvent utilisé pour afficher le numéro de page.

- **En-tête/Pied de groupe**

Dans un rapport on peut faire un regroupement d'enregistrements selon les valeurs d'un ou de plusieurs champs spécifiés (p.ex. Regrouper une liste de voitures par marque). Chaque groupe défini peut disposer d'un en-tête et d'un pied de groupe. L'en-tête de groupe affiche par exemple une ou plusieurs zones de texte indiquant le contenu du groupe (p.ex. Nom de la marque), ou les étiquettes de la section détail. Le pied de groupe contient des calculs (p.ex. sous totaux, moyennes) pour ce groupe. Entre l'en-tête de groupe et le pied de groupe se trouve la section détail, avec tous les enregistrements faisant partie du groupe.

- **Section Détail**

Cette section est la plus importante. Elle contient la plupart des zones de texte et affiche les données et les calculs pour chaque enregistrement. Il existe toujours une seule zone détail, indépendant du fait qu'il y a des groupes ou non.

## 9.2 Création d'un rapport

Voici quelques points de réflexion avant la création d'un rapport:

- Quelles données est-ce qu'on veut représenter ? (Dressez la liste des champs)
- Quelles informations supplémentaires sont utiles (p.ex. groupements, sous totaux, moyennes, pourcentages)
- Quel est le format approprié en terme de disposition des informations ?

**A faire : TP No 8**

**A faire : TP No 9**

# Partie 3 : Protection des données

# 10. Sécurité des données

## 10.1 Définition

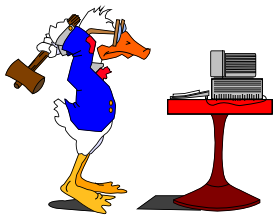
Par **sécurité des données**, on entend toutes les mesures prises pour que les données d'une BD soient protégées contre:

- les manipulations malveillantes
- les accès non autorisés;
- les incohérences et pertes de données accidentelles.

## 10.2 Les manipulations malveillantes

### 10.2.1 Définition

**Par manipulation malveillante, on entend la lecture, la modification ou la destruction non autorisée de données.**



**Exemple:**

Sachant qu'une BD est implémentée par un ou plusieurs fichiers, au niveau du système d'exploitation, une personne peut effacer une BD complète au niveau de la gestion des fichiers (p.ex. Explorer[1]) sans même avoir besoin de démarrer le SGBD.

**Exercice**

Donnez un exemple supplémentaire d'une manipulation malveillante.

[1] programme de gestion des fichiers sous Windows 95 et Windows NT

## 10.2.2 La protection contre les manipulations malveillantes

Il est difficile d'empêcher une personne autorisée dans le système à effectuer une manipulation malveillante.

 Toutefois, la plupart des SGBD exécutés sur un serveur offrent à l'administrateur d'une BD la possibilité de stocker toutes les manipulations effectuées dans une BD spécialisée, appelée journal des opérations effectuées (angl. auditing).

A l'intérieur du journal, l'administrateur peut à chaque moment vérifier quel utilisateur a effectué quelle manipulation sur quelle table à quel moment.

**Avantages:**

- Transparence totale concernant les manipulations effectuées.
- Identification des coupables en cas de problèmes.
- Le fait de rendre l'existence d'un tel journal public possède un certain effet psychologique sur les malfaiteurs potentiels.

**Désavantages:**

- Les conclusions tirées de la consultation d'un journal, sont à considérer avec précaution puisqu'un utilisateur en possession d'un mot de passe d'une autre personne peut effectuer des manipulations malveillantes sous l'identité de celle-ci.
- Les performances d'une BD peuvent être dégradées puisque pour chaque manipulation d'une table, une inscription dans le journal doit être effectuée.
- Un journal permet le contrôle total du travail des utilisateurs d'une BD.

# 10.3 Les accès non autorisés

## 10.3.1 Définition

**Par accès non autorisé à une BD on entend le fait qu'une personne lit, modifie, insère ou efface des données d'une BD sans avoir une autorisation préalable respectivement un accès électronique (Nom utilisateur & Mot de passe)**

## 10.3.2 La protection contre les accès non autorisés

Il existe un certain nombre de mesures de protection contre les accès non autorisés.

### *10.3.2.1 Mot de passe*

**Une BD peut être protégée par un mot de passe. L'utilisateur désirant travailler avec la BD; doit indiquer un mot de passe avant d'ouvrir celle-ci.**

Une fois la BD ouverte, l'utilisateur peut accéder à tous les objets.

**Avantage:**

Une personne ne disposant pas du mot de passe correspondant ne peut pas du tout accéder à une BD.

**Désavantage:**

Les mots de passe sont évidemment stockés dans un fichier spécial au niveau du système d'exploitation. Une personne ayant des connaissances approfondies d'un système d'exploitation n'a généralement aucun problème d'afficher le contenu d'un tel fichier. Pour cela, la plupart des SGBD utilisent un procédé d'encryptage afin de rendre les mots de passe illisibles avant de les stocker dans un fichier.

### *10.3.2.2 Droits d'accès aux objets d'une BD*

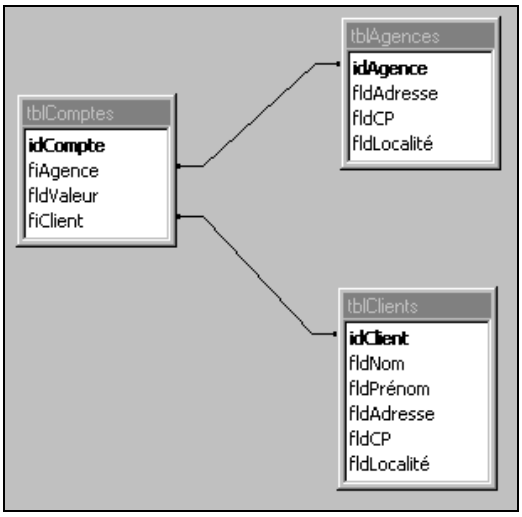
Au niveau des BD, qui se trouvent localement sur un PC, un mot de passe est généralement suffisant pour garantir une certaine sécurité. Par contre pour les BD, qui se trouvent sur un serveur géré par un administrateur[1], et qui sont accédées par une multitude d'utilisateurs, d'autres mécanismes plus variés s'imposent.

[1] personne (informaticien) responsable de la gestion du serveur, du SGBD sur le serveur et des BD

**Certains utilisateurs autorisés de la base peuvent être limités, dans leur accès, à quelques tables de celle-ci.**

**Exemple:**

Soit une BD pour la gestion des comptes d'une banque, implémentée sur un serveur BD, auquel tous les employés (même ceux des agences) ont un accès via un réseau informatique.




Un stagiaire auprès de la banque aura un login[1] afin d'accéder la base de données, mais l'administrateur de la base lui accorde uniquement un accès en lecture aux tables *tblAgences* et *tblClients*. En plus, l'administrateur crée une vue[2], qui contient tous les enregistrements de la table *tblComptes*, toutefois sans afficher le champ *fldValeur*.

Le stagiaire, avec les connaissances acquises pendant le cours d'informatique en classe de 13CG, peut créer sur son PC des requêtes, formulaires et rapports, mais il sera limité à l'utilisation des données pour lesquelles il est en possession des droits nécessaires.

En ce qui concerne les tables et vues d'une BD sur un serveur, l'administrateur n'a pas uniquement la possibilité de limiter les objets qu'un utilisateur peut accéder, mais il peut également définir pour chaque objet, le type d'accès auquel un utilisateur a le droit .

Parmi les types d'accès nous distinguons:

- **Autorisation de lecture** (angl. select)
  L'utilisateur peut uniquement lire des données.

- **Autorisation d'insertion** (angl. insert)
  L'utilisateur peut lire et insérer des données.
  Il ne peut cependant pas modifier ou effacer des données existantes.

[1] nom utilisateur & mot de passe à l'aide duquel un utilisateur peut s'identifier au système
[2] terme généralisé pour une requête de sélection stockée et réaffichable

- **Autorisation de mise à jour** (angl. update)
  L'utilisateur peut lire et modifier des données.
  Il ne peut cependant pas insérer ou effacer des données.

- **Autorisation d'effacement** (angl. delete)
  L'utilisateur peut lire et effacer des données.
  Il ne peut cependant pas insérer ou modifier des données.

Le SGBD sur le serveur garantit que les restrictions définies pour un utilisateur ne sont pas violées.

**Exemple:**

L'administrateur d'une BD gérée par un SGBD serveur Oracle peut par exemple exécuter des commandes comme:

```
GRANT insert, update ON tblComptes,tblAgences TO JWEBER;
```

Cette commande donne à l'utilisateur identifié au système par le nom JWEBER, le droit de lire les données des tables *tblComptes* et *tblAgences*, d'insérer de nouveaux enregistrements dans ces tables et de modifier les enregistrements existants dans les deux tables.

La commande suivante enlève le droit d'insertion dans la table *tblComptes* à l'utilisateur.

```
REVOKE insert ON tblComptes FROM JWEBER;
```

**Exercice**

En vous référant à la syntaxe présentée dans cet exemple, et en supposant que le nom utilisateur du stagiaire de l'exemple précédent est EMULLER, indiquez les commandes nécessaires pour donner les droits d'accès au stagiaire de la banque au début de la période de stage, et celles nécessaires pour lui enlever ces droits à la fin de la période de stage. Nous supposons que la vue créée par l'administrateur s'appelle *vComptesSansValeurs*.

---

**Avantage de la gestion des droits d'accès:**

Les droits d'accès sont un outil parfait pour personnaliser l'accès à une BD de façon à ce que chaque utilisateur puisse uniquement effectuer les opérations en relation avec sa fonction et compétence à l'intérieur de l'entreprise. Ceci restreint les possibilités d'effectuer des manipulations malveillantes et limite en plus le nombre des suspects en cas d'une telle manipulation.

**Désavantage de la gestion des droits d'accès:**

En fait, il n'existe pas vraiment un désavantage, mais la gestion des droits d'accès nécessite un effort de gestion supplémentaire considérable, surtout pour les sociétés où les compétences des employés varient beaucoup.

### *10.3.2.3 Sécurisation du système d'exploitation*

Un SGBD, tout comme les autres applications informatiques, utilise les services d'un système d'exploitation.

Une BD est toujours implémentée à l'aide de un ou de plusieurs fichiers. Le contenu de ces fichiers est normalement illisible pour chaque application outre que le SGBD à l'aide duquel le fichier (la BD) a été créé.

Toutefois, il est possible d'endommager et même d'effacer complètement un tel fichier, ce qui aurait comme conséquence la destruction partielle ou totale de la BD, de façon indépendante des mécanismes de sécurité implémentés au niveau du SGBD.

 **Il convient donc de protéger même l'accès au système d'exploitation, c.à.d. l'accès général au PC par un mot de passe.**

Au niveau d'un PC, qui contient une BD locale, la plupart des systèmes d'exploitation prévoient deux types de mot de passe:

1. Un mot de passe pour démarrer le PC (angl. Power On Password)
2. Un mot de passe couplé à un économiseur d'écran[1] (angl. Screen Saver Password)

Pour les serveurs BD, quelques mesures supplémentaires, telles que l'emplacement dans une salle protégée par une clé électronique, s'imposent.

**Avantages:**

L'existence d'un mot de passe au niveau du système d'exploitation augmente le niveau de sécurité du système.

**Désavantage:**

Un utilisateur doit indiquer son nom d'utilisateur ainsi que son mot de passe deux fois, la première fois pour accéder au système d'exploitation et la deuxième fois pour accéder à la BD à l'aide du SGBD. Certains SGBD sont cependant capables de reconnaître le nom d'utilisateur ainsi que le mot de passe indiqué au système d'exploitation et de le reprendre lorsque l'utilisateur veut accéder à une BD.

[1] programme affichant une animation à l'écran, qui s'exécute automatiquement après un nombre prédéfini de minutes sans activité de l'utilisateur

# 10.4 Les incohérences et pertes de données accidentelles

## 10.4.1 Définition

**Par incohérence accidentelle, on entend toute coupure non intentionnelle des liens logiques entre les données d'une BD.**

**Exemple d'une incohérence accidentelle:**

Dans les systèmes multi-utilisateur, il se peut que deux utilisateurs accèdent en même temps, aux mêmes enregistrements d'une BD sur le serveur. On parle d'un accès concurrent.


Nous supposons, que les deux utilisateurs exécutent en même temps, de façon indépendante l'un de l'autre, les deux requêtes suivantes:

| Utilisateur 1 | Utilisateur 2 |
|---|---|
| `UPDATE Employés`<br>`SET fldDépartement="CPT"`<br>`WHERE fldDépartement="Comptabilité";` | `UPDATE Employés`<br>`SET fldSalaire=fldSalaire*1.1`<br>`WHERE fldDépartement="Comptabilité"`<br>`  AND fldDateNaiss<#1/1/70#;` |
| Nouvelle codification pour le service de comptabilité. | Tous les employés du service comptabilité nés avant le 1/1/70 , subissent une hausse de salaire de 10%. |

Nous supposons en plus que la requête de l'utilisateur numéro 2 est exécutée quelques instants avant l'autre requête.

Cependant, la requête de l'utilisateur 1 s'exécute un peu plus vite que l'autre, puisque pour chaque enregistrement, il y a uniquement un seul critère de sélection à vérifier.

A un certain moment, l'exécution de la requête 1 aura dépassée celle de la requête 2, donc pour certains enregistrements, le code "Comptabilité" est changé en "CPT", avant que la

requête 2 ne puisse effectuer la modification du salaire. Parmi tous les employés ayant droit à une hausse de salaire, certains sont donc "ignorés".

Le problème des accès concurrents se pose surtout dans les systèmes avec beaucoup d'utilisateurs émettant beaucoup de requêtes, tels que par exemple la gestion des dépôts d'une banque.

Ce problème peut être résolu par le mécanisme de la sérialisation d'exécution des requêtes[1], supporté automatiquement par tous les SGBD multi-utilisateur exécutés sur un serveur de BD. Ce mécanisme garantit une exécution en série de plusieurs requêtes, même lorsque celles-ci sont envoyées par plusieurs utilisateurs en même temps.

**Exercice**

Donnez un exemple supplémentaire d'une incohérence accidentelle et d'une perte accidentelle.

## 10.4.2 La protection contre les incohérences et pertes de données accidentelles

Tous les SGBD implémentent des fonctionnalités, qui garantissent la cohérence des données en fonctionnement normal. A titre d'exemple mentionnons les contraintes d'intégrité et la sérialisation d'exécution des requêtes.

Une incohérence accidentelle peut donc en principe uniquement apparaître suite à une perte accidentelle de données. Citons la perte d'enregistrements, qui contiennent des clés primaires liées à des clés étrangères d'une autre table.

Par conséquent, nous allons limiter la discussion suivante aux pertes accidentelles.

Les causes des pertes de données accidentelles sont réparties en trois groupes:

1. Les pertes provoquées par des erreurs humains;
2. Les pertes des données en mémoire interne (RAM).
3. Les pertes des données stockées sur disque dur.

[1] une requête n'est exécutée qu'au moment où la requête précédente a terminé son exécution

### 10.4.2.1 Les pertes provoquées par des erreurs humaines

Ce type de pertes est difficilement maîtrisable. Toutefois, une bonne formation des utilisateurs d'un système aide à réduire le nombre de telles pannes.

**Exemple d'une perte provoquée par une erreur humaine:**

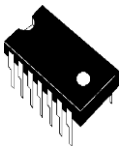
Une requête de suppression mal formulée en SQL, efface trop d'enregistrements.

### 10.4.2.2 Les pertes des données en mémoire interne (RAM)

Les BD résidant normalement sous forme de fichier(s) sur le disque dur, sont partitionnées en blocs de longueur fixe, avec chaque bloc contenant un ou plusieurs enregistrements. Un SGBD appelle les blocs nécessaires en mémoire centrale et les retourne sur le disque suite aux modifications effectuées.

Les données résidant en mémoire interne ne résistent pas à un crash; et sont perdues de façon irrécupérable. Comme les blocs avec les enregistrements sont généralement retournés vers le disque dur assez vite après une opération de modification, l'impact d'une perte de données en mémoire interne n'est toutefois pas très grand.

**Exemples de causes pour la perte de données en mémoire interne:**



- un crash système provoqué par un défaut matériel;
- un crash système provoqué par un défaut logiciel;
- une coupure d'alimentation électrique.

### 10.4.2.3 Les pertes des données stockées sur disque dur

Les données stockées sous forme de fichier(s) sur disque dur peuvent en principe également être perdues, mais heureusement il existe des mesures de prévention d'une perte de données sur disque, puisque l'impact d'une telle perte peut être énorme, et peut dans le pire, aboutir dans la perte complète de la BD.

**Exemples de causes pour la perte de données sur disque dur:**

- une manipulation erronée effectuée par un utilisateur;
- une erreur de logiciel (angl. Bug) a causée une incohérence de certaines données;
- une panne d'un disque.

## 10.4.3 Les mesures de prévention contre la perte de données

### *10.4.3.1 La sauvegarde des données (angl. backup)*

Une méthode préventive contre la perte de données sur disque dur est la sauvegarde régulière des données du (des) disque(s).

**L'opération de sauvegarde (ang. backup)** d'une BD consiste dans la copie du resp. des fichiers qui contiennent la BD, du disque dur vers un support de sauvegarde. Ceci est fait au niveau du système d'exploitation.
Lors d'une perte de données d'un disque, on peut **restituer (angl. restore)** les données sur le disque à partir du support de sauvegarde.

Afin de pouvoir effectuer une sauvegarde des fichiers BD au niveau du système d'exploitation, la BD doit être "fermée", ce qui veut dire que personne ne doit être en train d'effectuer n'importe quelle manipulation. Sinon, on risque de sauvegarder des fichiers incohérents.

Au niveau des SGBD pour serveurs, il existe des systèmes, qui permettent la sauvegarde "intelligente" d'une table pendant que celle-ci est consultée ou même modifiée.

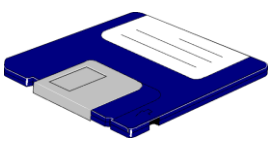
On distingue généralement deux types de sauvegardes:

1. La **sauvegarde complète** (angl. full backup)
   Toutes les données sont sauvegardées

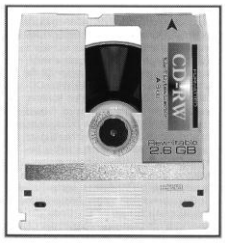
2. La **sauvegarde incrémentale** (angl. incremental backup)
   Uniquement les nouvelles données ou celles modifiées depuis la dernière sauvegarde sont sauvegardées.

Il est conseillé de gérer plusieurs **générations de sauvegarde**. On aura ainsi une version "Lundi", "Mardi", "Mercredi" etc., afin de pouvoir accéder à un état de données antérieur si la dernière version sauvegardée est déjà corrompue.

Voici un tableau comparatif des supports de sauvegarde, qui sont actuellement assez répandus.

| Support | Caractéristiques |
|---|---|
| Disquette  | • Stockage magnétique<br>• Capacité[1]: 720KB/1.4MB (très faible)<br>• Lecture & écriture répétitive des données<br>• Vitesse d'accès assez lente |

[1] 1KB = 1Kilobyte = 1024 Byte / 1MB = 1 Megabyte = 1024KB / 1GB = 1Gigabyte = 1024MB

| | |
|---|---|
| Bande magnétique<br>  | • Stockage magnétique<br>• Capacité: 1GB – 40 GB (très élevée)<br>• Lecture & écriture répétitive des données<br>• Vitesse d'accès très lente |
| Disque magnétique amovible<br>  | • Stockage magnétique<br>• Capacité: 100MB – 2GB (suffisante pour petits systèmes)<br>• Lecture & écriture répétitive des données<br>• Vitesse d'accès rapide |
| CD-R<br>  | • Stockage optique<br>• Capacité: 650 MB - 2GB (suffisante pour petits systèmes)<br>• Lecture répétitive & une seule écriture<br>• Vitesse d'accès moyenne |
| CD-RW<br>  | • Stockage magnéto-optique<br>• Capacité: 1 – 3 GB (suffisante pour petits systèmes)<br>• Lecture & écriture répétitive des données<br>• Vitesse d'accès moyenne |

### 10.4.3.2 La réplication du disque dur (angl. mirroring)

Cette solution met en œuvre plusieurs disques durs dont le contenu est identique. La gestion incombe au système d'exploitation resp. à un contrôleur de disque (carte électronique). Celui-ci doit donc s'assurer que les informations sur les différents disques soient à jour en permanence, de façon à ce que l'on puisse continuer à travailler en cas de panne d'un disque.

### 10.4.3.3 Réplication du serveur (angl. Backup server)

Dans cette solution, le serveur de réseau (qui peut contenir des données + le système d'exploitation) est répliqué (dédoublé). Si le serveur principal a une défaillance, on continue à travailler sur le serveur de sauvegarde.

### 10.4.3.4 Les systèmes RAID-5

Le terme RAID (angl. Redundant Array of Inexpensive Disks) dénote un système dans lequel plusieurs disques durs sont gérés par un contrôleur spécifique qui répartit les données de telle façon sur les disques de manière à ce que l'on puisse échanger l'un des disques sans qu'il y ait perte de données. Les dernières versions de contrôleur permettent même le 'hot-swapping', c.-à-d. l'échange d'un disque défectueux sans arrêter le système.

# Partie 4 : Travaux sur logiciel

# 11. Travaux sur logiciel

## 11.1  TP No 1 : Introduction à MS-Access

DUREE APPROX. : 2h.

**PRESENTATION**

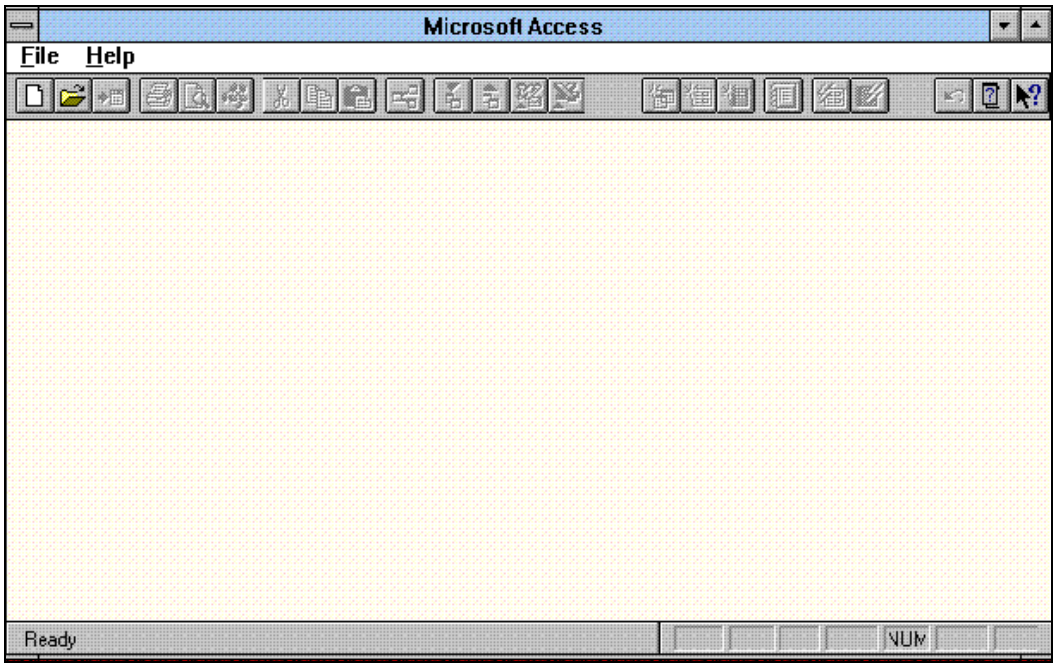
Pendant ce cours nous allons utiliser le SGBD **MS-Access**, qui fait partie du groupe de programmes **MS-Office** en version professionnelle. (Attention: La version standard de MS-Office ne contient pas MS-Access).

Une base de données MS-Access est stockée dans un fichier qui porte l'extension **.mdb** . Ce fichier contient tous les objets définis pour cette base.

**MANIPULATIONS IMPORTANTES**

| ACTION | COMMENT FAIRE ? |
|---|---|
| Démarrer MS-Access | Sélectionner l'icône :  Microsoft Access |
| Créer une nouvelle BD | |
| Ouvrir une BD existante | |
| Afficher le contenu du système d'aide en ligne | |
| Effectuer une recherche par mot clé dans le système d'aide en ligne | |
| Recevoir de l'aide contextuelle lors d'une manipulation | |
| Recevoir de l'aide concernant une option de menu ou une icône | |
| Quitter MS-Access | |
| | |

L'écran suivant se présente comme écran principal de MS-Access:



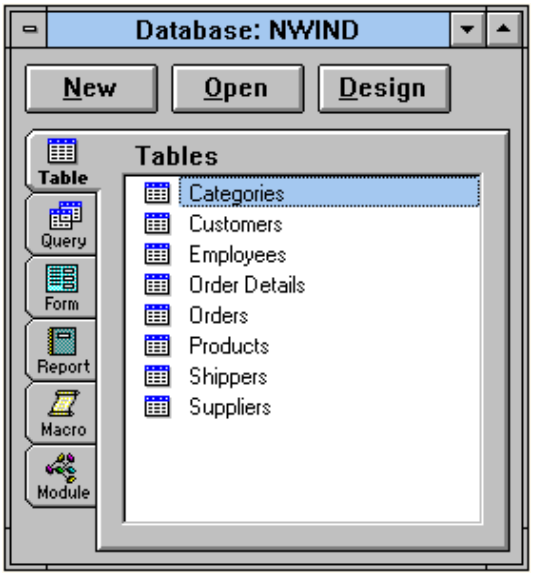

**ACTIVITES**

1. Démarrez MS-Access

2. Essayez de localiser un fichier .mdb existant sur votre disque dur. En principe, MS-Access contient une BD exemple *nwind.mdb*, qui est normalement stockée dans le répertoire .....*access**sampapps*.

3. Ouvrez la base de données sous MS-Access.

   Lorsque vous avez ouvert une BD, la **fenêtre des objets de la BD** apparaît:



Nous voyons que MS-Access connaît entre autres les 4 types d'objets que nous venons d'introduire:

| Français | Anglais |
|---|---|
| Table | Table |
| Requête | Query |
| Formulaire | Form |
| Rapport | Report |

**Pour chaque objet, il existe deux modes:**

- **Le mode Utilisation de l'objet → Utiliser l'objet**
- **Le mode Structure de l'objet → Créer l'objet resp. modifier sa structure**

4. Démarrez le conseiller de MS-Access

5. Sélectionnez l'option "See a quick overview"

6. Affichez et lisez attentivement le contenu des éléments Databases, Tables, Queries, Forms et Reports. Répondez aux questions suivantes:

7. En ce qui concerne les tables, quelle est la différence entre "Design View" et "Datasheet View", et quelle est la relation avec les deux modes (Mode Utilisation, Mode Structure) que nous venons de définir ?

8. Pour les autres objets (Requêtes, Formulaires, Rapports), quelles sont les dénominations des modes "Structure" et "Utilisation" dans le langage de MS-Access.

| Objet | Mode Structure | Mode Utilisation |
|---|---|---|
| **Table** | Design View | Datasheet View |
| **Requête** | | |
| **Formulaire** | | |
| **Rapport** | | |

9. Décrivez en une seule phrase l'utilité des objets suivants:

| | |
|---|---|
| **Table** | |
| **Requête** | |

| | |
|---|---|
| **Formulaire** | |
| **Rapport** | |

10. Fermez le conseiller.

11. Quittez MS-Access

# 11.2  TP No 2 : Les tables

DUREE APPROX : 3h.

**PRESENTATION**

Dans le cadre de ce TP, nous allons créer une table et la remplir avec des données. En plus, nous allons examiner un peu les possibilités offertes par Access en ce qui concerne la manipulation des données dans une table, ainsi que l'importation et l'exportation de données.

Le mode "Design View" de MS-Access correspond au mode "Structure" (ou mode "Création") qu'on vient de définir (voir TP1). Ceci veut dire qu'on peut définir la structure de la table (champs, clé primaire …).
Le mode "Datasheet View" de MS-Access correspond au mode "Utilisation" qu'on vient de définir. Ceci veut dire qu'on peut ajouter, modifier et supprimer des enregistrements dans la table.

**MANIPULATIONS IMPORTANTES**

Définir la structure d'une table:

| ACTION | COMMENT FAIRE ? |
|---|---|
| Créer une table (Sans utiliser l'assistant) | |
| Ouvrir une table existante en mode "Création" | |
| Définir un champ | |
| Déplacer un champ (Modifier l'ordre des champs) | |
| Insérer un champ | |
| Supprimer un champ | |
| Définir une clé primaire à un champ | |
| Définir une clé primaire multi-champs | |
| Créer un index sur un seul champ | |
| Créer un index multi-champs | |
| Imprimer la structure de la table | |
| Enregistrer la table | |
| | |

Utiliser une table/Travailler avec les enregistrements:

| ACTION | COMMENT FAIRE ? |
|---|---|
| Sélectionner le mode "Feuille de Données" pour une table ouverte en mode "Création" | |
| Ouvrir une table en mode "Feuille de Données" | |
| Déplacer le curseur dans le champ suivant | |
| Déplacer le curseur dans le champ précédent | |
| Déplacer le curseur dans n'importe quel champ d'un enregistrement | |
| Sélectionner l'enregistrement suivant | |
| Sélectionner l'enregistrement précédent | |
| Sélectionner le premier enregistrement | |
| Sélectionner le dernier enregistrement | |
| Comment reconnaît-on l'enregistrement en cours ? | |
| Modifier la largeur d'une colonne | |
| Déplacer un champ | |
| Rechercher des valeurs dans les enregistrements (angl. Find) | |
| Masquer des champs (angl. Hide) | |
| Réafficher les champs masqués (angl. Unhide) | |
| Imprimer des enregistrements | |
| Enregistrer les ajouts resp. modifications | |
| | |

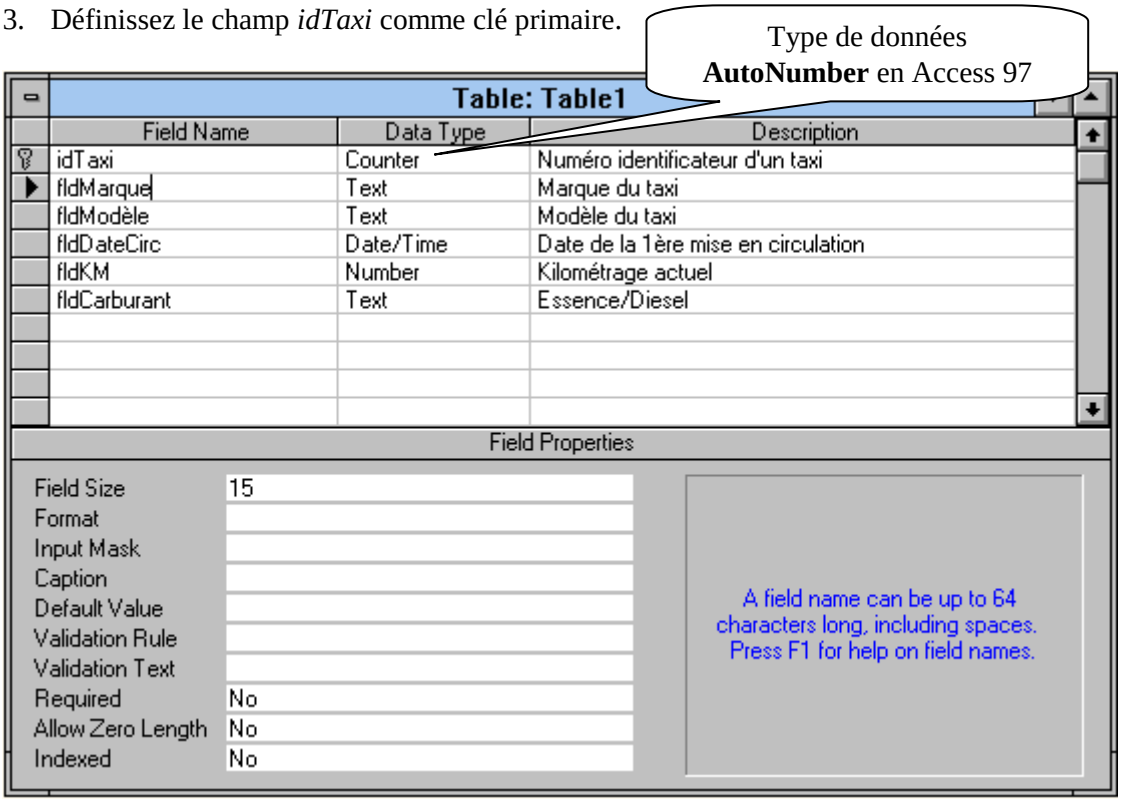
**ACTIVITES**

1. Créez une nouvelle BD que vous appelez *taxis.mdb* .

2. Créez une table *tblTaxis* avec les champs suivants. N'utilisez pas l'assistant (angl. Wizard)

| **Nom du champ** | **Type de données** | **Description** |
|---|---|---|
| idTaxi | Compteur | Numéro identificateur d'un taxi |
| fldMarque | Texte [15] | Marque du taxi |
| fldModèle | Texte [15] | Modèle du taxi |
| fldDateCirc | Date | Date de la première mise en circulation |
| fldKm | Numérique | Kilométrage actuel |
| fldCarburant | Texte [10] | Essence/Diesel |

3. Définissez le champ *idTaxi* comme clé primaire.




4. Expliquez brièvement l'utilité du type de données "Compteur" (angl. Counter).

5. Sauvegardez la définition de la table (Nom = *tblTaxis*).

6. Entrez les enregistrements suivants dans votre table

| **idTaxi** | **fldMarque** | **fldModèle** | **fldDateCirc** | **fldKM** | **fldCarburant** |
|---|---|---|---|---|---|
| 1 | BMW | 520i | 01.02.1995 | 85000 | Essence |
| 2 | Ford | Scorpio | 15.07.1994 | 110800 | Diesel |

| | | | | | |
|---|---|---|---|---|---|
| 3 | VW | Passat | 28.09.1996 | 56700 | Diesel |
| 4 | BMW | 525i | 10.10.1996 | 48500 | Essence |
| 5 | Audi | A6 | 09.11.1997 | 4500 | Diesel |
| 6 | VW | Golf TDI | 23.09.1997 | 7800 | Diesel |
| 7 | VW | Passat | 22.02.1996 | 59000 | Diesel |

7. Créez un index multi-champs, que vous appelez simplement *Ind1*, sur la marque, le modèle et la date de mise en circulation. A l'intérieur de l'index, les trois champs sont triés par ordre ascendant.

8. Ajoutez l'enregistrement suivant

| | | | | | |
|---|---|---|---|---|---|
| 8 | VW | Passat | 13.10.1994 | 100800 | Diesel |

9. Sauvegardez et fermez la table.

# 11.3  TP No 3 : Les requêtes portant sur une table

DUREE APPROX : 4h.

**PRESENTATION**

Ce TP contient une partie théorique supplémentaire au cours. Cette partie nous fournit à l'aide d'exemples un aperçu sur les possibilités de réaliser des requêtes à l'aide de Access-QBE. Chaque exemple nous présente une requête SQL ainsi que son pendant en Access-QBE. Le mécanisme des requêtes paramétrées est également présenté.

MS Access nous offre pour chaque requête une transparence totale entre les spécifications QBE et le code SQL correspondant. L'utilisateur qui travaille en QBE peut à tout moment changer en vue SQL, qui lui permet de visualiser le code SQL correspondant à la spécification QBE actuelle. Il peut alors ajouter respectivement modifier le code SQL . Les modifications seront répercutées dans la grille QBE. Access nous offre donc une synchronisation complète dans les deux sens.

Il est donc important de savoir que Access supporte deux modes de création pour les requêtes:
1. Le mode QBE
2. Le mode SQL

Access ne connaît pas seulement les requêtes de sélection, mais également les requêtes de modification de suppression et d'insertion. Les trois derniers types de requêtes sont dénommées généralement *Requêtes Action (angl. Action Queries)*.

Toutefois, il existe des requêtes, qui ne peuvent pas être spécifiées via QBE. Parmi celles-ci se trouvent les requêtes d'insertion telles que nous les avons vues en SQL.

```
INSERT INTO <Nom de la table> [<Nom des champs>]
VALUES ( <Valeurs pour les champs> );
```

Lorsqu'on veut insérer des enregistrements, un à la fois, on utilise SQL ou on insère directement dans la table.

Chaque nouvelle requête est par défaut une requête de sélection. L'utilisateur doit explicitement spécifier le type de requête pour les autres types de requête. Un utilisateur expérimenté peut directement spécifier que la requête est une requête de suppression ou de modification, et entrer les critères de sélection.

**Une manière plus sûre de procéder est la suivante:**
1. **Définir une requête de sélection dont les critères de sélection correspondent exactement à ceux de la requête action.**
2. **Vérifier le résultat de la requête sélection.**
3. **Convertir la requête sélection en requête action et exécuter la requête action.**

Voici une table récapitulative des requêtes possibles en QBE et en SQL.

| MS Access | SQL | QBE |
|---|---|---|
| Requête de sélection | Oui | Oui |
| Requête d'insertion | Oui | Non[1] |
| Requête de suppression | Oui | Oui |
| Requête de modification | Oui | Oui |

Il est possible de sauvegarder une requête (Sauvegarde de la spécification QBE resp. du code SQL) et de l'exécuter à un moment ultérieur.

**Convention des noms:**

Les noms des requêtes sont précédés des préfixes suivants:
1. Requêtes de sélection → **qsel** (angl.: Query Select)
2. Requêtes d'insertion → **qins** (angl.: Query Insert)
3. Requêtes de modification → **qupd** (angl.: Query Update)
4. Requêtes de suppression → **qdel** (angl.: Query Delete)

**MANIPULATIONS IMPORTANTES**

| ACTION | COMMENT FAIRE ? |
|---|---|
| Créer une nouvelle requête (Sans utiliser l'assistant) | |
| Ouvrir une requête existante en mode création | |
| Changer en mode SQL | |
| Changer en mode QBE | |
| Changer en mode Feuille de Données (Affichage du résultat) | |
| Sauvegarder une requête | |
| Sélectionner une table en mode QBE | |
| Sélectionner les champs en mode QBE | |
| Désélectionner des champs en mode QBE | |
| Activer en mode QBE l'affichage d'un champ sélectionné (Correspond à la clause SELECT en SQL) | |

[1] Access supporte cependant la syntaxe **INSERT INTO <Table cible> SELECT ...** , qui copie les enregistrements résultant de l'exécution du SELECT dans la table cible.

## THEORIE SUPPLEMENTAIRE - IMPLEMENTATION DE ACCESS-QBE

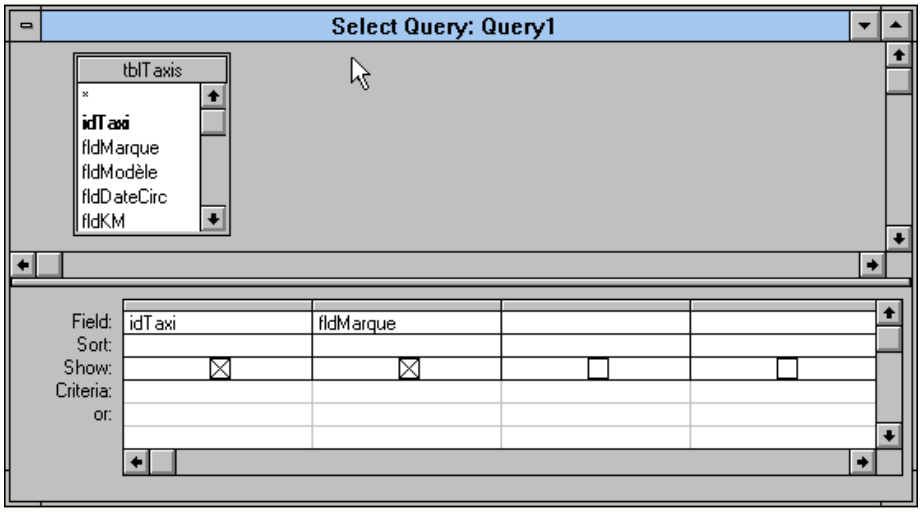
Les exemples suivants sont basés sur la table *tblTaxis* , créée lors du **TP2.**

| **Nom du champ** | **Type de données** | **Description** |
|---|---|---|
| **idTaxi** | Compteur | Numéro identificateur d'un taxi |
| fldMarque | Texte [15] | Marque du taxi |
| fldModèle | Texte [15] | Modèle du taxi |
| fldDateCirc | Date | Date de la 1ère mise en circulation |
| fldKm | Numérique | Kilométrage actuel |
| fldCarburant | Texte [10] | Essence/Diesel |

Voici à titre d'exemple un écran QBE qui représente la requête SQL suivante.

- Affichez le numéro et la marque de tous les taxis

```
SELECT idTaxi, fldMarque
FROM tblTaxis;
```




Requêtes de sélection

Afin d'afficher un champ, ce champ doit donc être sélectionné et l'affichage du champ doit être activé.
Le fait de sélectionner l'astérisque ( * ) correspond à la sélection de tous les champs de la table. (Correspond à `SELECT * FROM ...` )

L'option DISTINCT:

Afin d'implémenter l'option **DISTINCT** en Access-QBE la propriété *Unique Values* de la requête doit avoir la valeur **YES**. Pour afficher les propriétés d'une requête, cliquez sur la fenêtre QBE avec la touche droite de la souris. Sélectionnez ensuite l'option **Properties** du menu flottant.

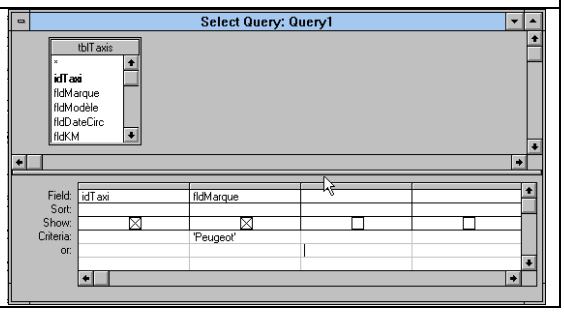
Les critères de sélection

En principe , Access-QBE prévoit la ligne intitulée *Criteria* pour définir les critères de sélection

Affichez le numéro et la marque pour tous les taxis de la marque "Peugeot".

```
SELECT idTaxi, fldMarque
FROM tblTaxis
WHERE fldMarque='Peugeot';
```

=



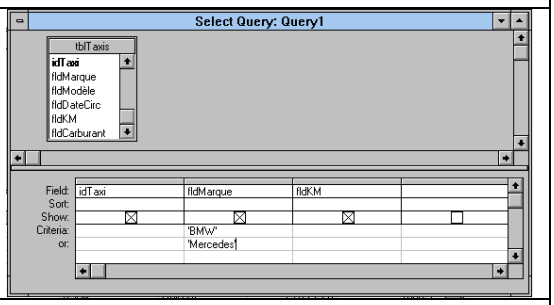

Les opérateurs AND et OR

- Lorsqu'on spécifie plusieurs critères de sélection pour plusieurs champs différents sur une seule ligne, ces critères seront combinés par un ET (AND) logique. Alternativement il est possible d'utiliser l'opérateur AND directement dans une même cellule.
- Lorsqu'on spécifie plusieurs critères dans plusieurs lignes l'une au-dessous de l'autre, pour un seul champ, ces critères seront combinés par un OU (OR) logique. Alternativement, il est possible d'utiliser l'opérateur OR directement dans une même cellule.

Affichez le numéro, la marque et le kilométrage des taxis où la marque est "BMW" ou "Mercedes".

```
SELECT idTaxi, fldMarque, fldKM
FROM tblTaxis
WHERE fldMarque = 'BMW' OR
fldMarque='Mercedes';
```

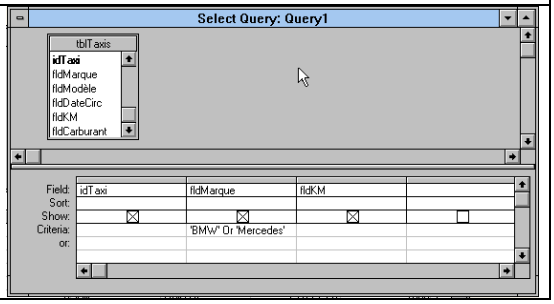
=




Même requête.

```
SELECT idTaxi, fldMarque, fldKM
FROM tblTaxis
WHERE fldMarque = 'BMW' OR
   fldMarque='Mercedes';
```

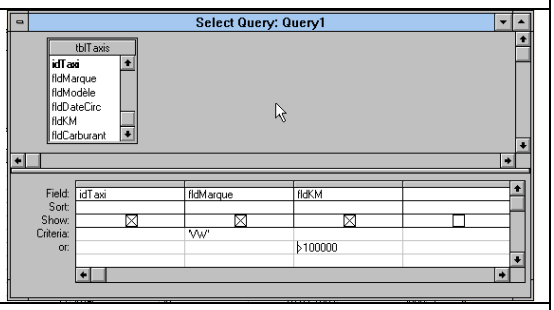
=



Affichez le numéro, la marque et le kilométrage en tenant uniquement compte des taxis qui sont des "VW" ou de ceux qui ont un kilométrage >100000.

```
SELECT idTaxi, fldMarque, fldKM
FROM tblTaxis
WHERE fldMarque='VW' OR fldKM
   >100000;
```

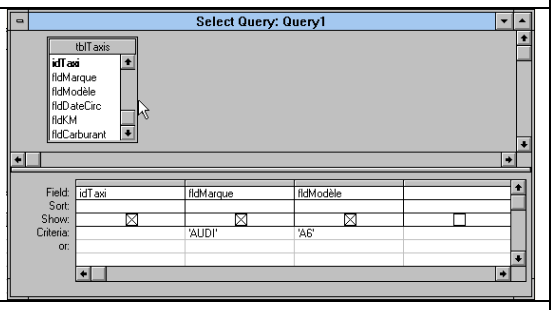
=




Affichez le numéro, la marque et le modèle des Audi A6.

```
SELECT idTaxi, fldMarque, fldModèle
FROM tblTaxis
WHERE fldMarque='AUDI' AND
   fldModèle='A6';
```

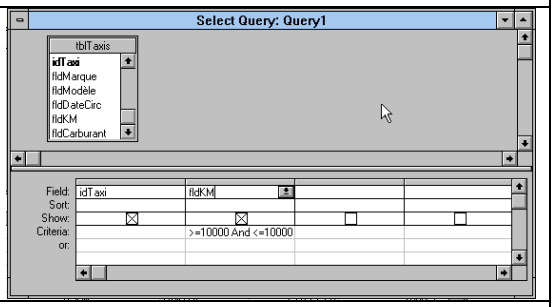
=




Affichez le numéro et le kilométrage des taxis ayant un kilométrage entre 10000km et 100000km, les deux valeurs incluses.

```
SELECT idTaxi, fldKM
FROM tblTaxis
WHERE fldKM >= 10000 AND fldKM
   <=100000;
```

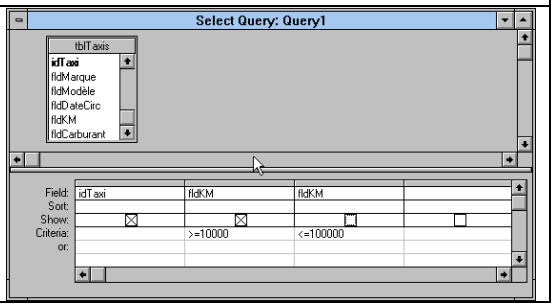
=




Même requête.

```
SELECT idTaxi, fldKM
FROM tblTaxis
WHERE fldKM >= 10000 AND fldKM
   <=100000;
```

=

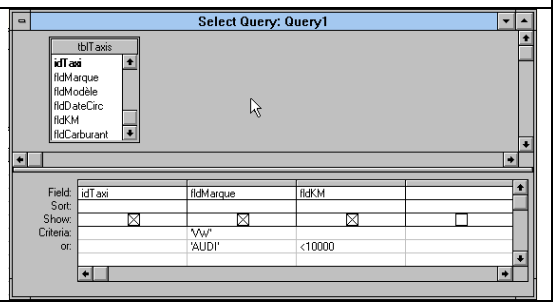


Affichez le numéro, la marque et le kilométrage en retenant uniquement soit les VW, soit les Audi ayant un kilométrage < 10000km.

```
SELECT idTaxi, fldMarque, fldKM
FROM tblTaxis
WHERE fldMarque = 'VW' OR fldMarque =
   'AUDI' AND fldKM < 10000;
```

(Attention: AND a la priorité sur OR)

=



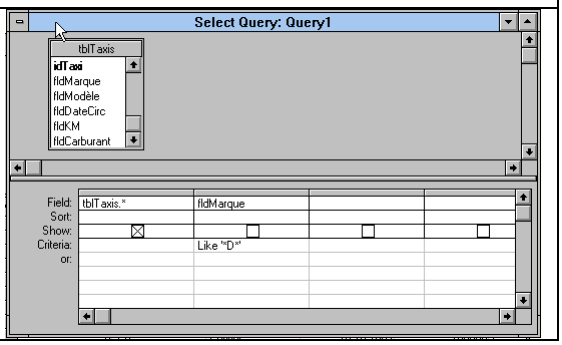

Les opérateurs LIKE, IN, IS NULL et BETWEEN

Ces opérateurs sont utilisés à l'intérieur des cellules, en respectant exactement la syntaxe de SQL.

Affichez toutes les informations disponibles pour les taxis dont le nom de marque contient la lettre "D".

```
SELECT *
FROM tblTaxis
WHERE fldMarque LIKE '*D*';
```

=



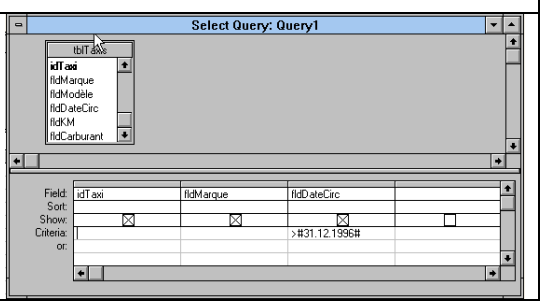

Les dates en Access-QBE

Access convertit les dates spécifiées en QBE automatiquement dans le format américain (mois/jour/année) lors d'une conversion en SQL.

Affichez le numéro, la marque et la date de première mise en circulation pour tous les taxis immatriculés après le 31/12/96.

```
SELECT idTaxi, fldMarque, fldDateCirc
FROM tblTaxis
WHERE fldDateCirc > #12/31/96#;
```

=



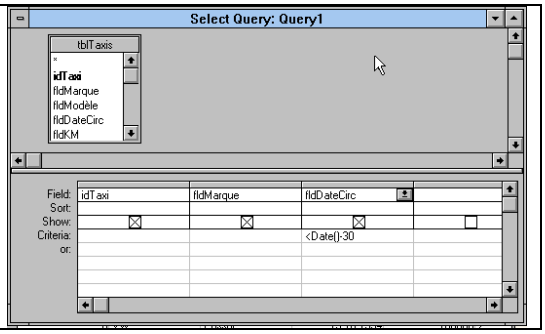
Access nous offre quelques fonctions supplémentaires à la syntaxe SQL. Ces fonctions peuvent être employées en QBE tant qu'en SQL.

| Fonction | Description |
|---|---|
| **Date()** | Retourne la date actuelle (Date système de l'ordinateur).<br>Le format dépend du format sélectionné dans Windows (c.f. Regional Settings) |
| **Year(**<date>**)**<br>**Month(**<date>**)**<br>**Day(**<date>**)** | Retourne l'année (100-9999) resp. le mois (1-12) resp. le jour (1-31) de la date qui est passée comme paramètre. |
| **Time()** | Retourne l'heure actuelle (Heure système de l'ordinateur).<br>Le format dépend du format sélectionné dans Windows (c.f. Regional Settings) |
| **Hour(**<time>**)**<br>**Minute(**<time>**)**<br>**Second(**<time>**)** | Retourne l'heure (0-23) resp. la minute (0-59) resp. la seconde (0-59) de l'heure, qui est passée comme paramètre. |

Il est possible de calculer avec les dates. L'expression **< Date() - 30** par exemple sélectionne toutes les dates antérieures à la date actuelle moins 30 jours.

```
SELECT idTaxi, fldMarque, fldDateCirc
FROM tblTaxis
WHERE fldDateCirc < Date()-30;
```

=



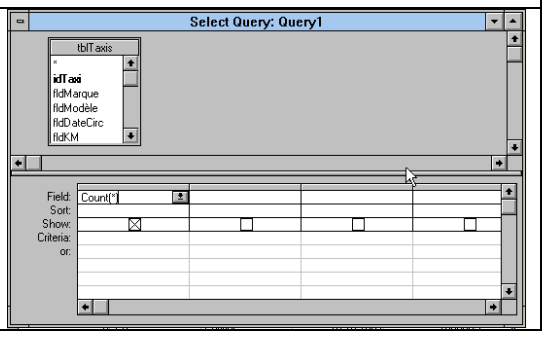

Les fonctions d'agrégation

Ces fonctions sont utilisées à l'intérieur des cellules, en respectant exactement la syntaxe de SQL.

Affichez le nombre de taxis enregistrés dans la table *tblTaxis*.

```
SELECT COUNT(*)
FROM tblTaxis
```

=



La clause GROUP BY

La clause GROUP BY est représentée en Access-QBE par une option des requêtes de sélection, les requêtes du type sous-total (ang. Totals Queries).

En fait, il suffit de sélectionner l'option de menu _____ resp. l'icône  afin d'insérer une nouvelle ligne dans votre écran QBE. Cette ligne, qui porte le libellé **Total:** , vous permet d'indiquer les champs de groupe.

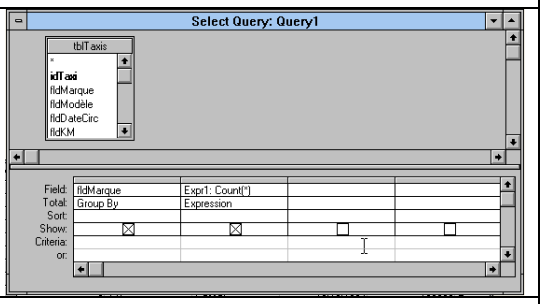
Une cellule appartenant à cette ligne peut prendre en principe 3 valeurs possibles.

- **Group By** Indique que le champ est un champ de groupe.
- **Expression** Pour les champs sur lesquels sont définis des fonctions d'agrégation. On peut directement sélectionner la fonction d'agrégation dans cette cellule (Exception: COUNT(*) ).
- **Where** Indique les champs, qui apparaissent dans le critère de sélection. Ces champs sont d'office non affichés.

Affichez pour chaque marque, le nom de la marque et le nombre de taxis pour cette marque.

```
SELECT fldMarque, COUNT(*)
FROM tblTaxis
GROUP BY fldMarque;
```

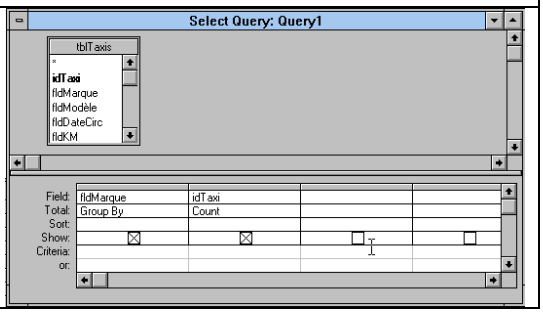
=




Même requête.

```
SELECT fldMarque, COUNT(idTaxi)
FROM tblTaxis
GROUP BY fldMarque;
```

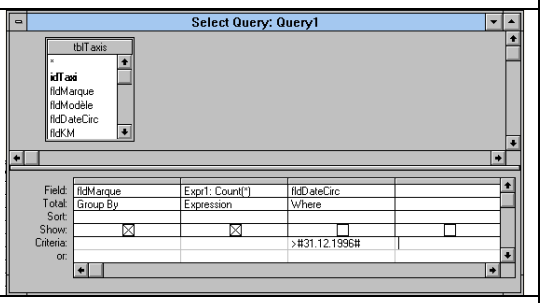
=



Même requête, en retenant uniquement les taxis immatriculés après le 31/12/96.

```
SELECT fldMarque, COUNT(*)
FROM tblTaxis
WHERE fldDateCirc > #12/31/96#
GROUP BY fldMarque;
```

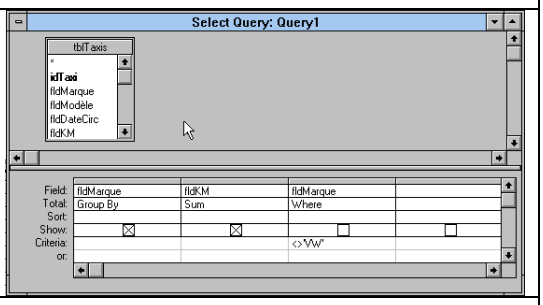
=




En ignorant les VW, affichez pour chaque marque, le nom de la marque ainsi que la somme du kilométrage parcouru.

```
SELECT fldMarque, SUM(fldKM)
FROM tblTaxis
WHERE fldMarque <> 'VW'
GROUP BY fldMarque;
```

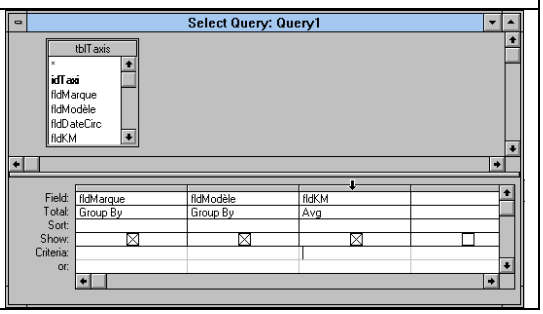
=




Affichez pour chaque combinaison marque/modèle la moyenne du kilométrage parcouru.

```
SELECT fldMarque, fldModèle,
   AVG(fldKM)
FROM tblTaxis
GROUP BY fldMarque, fldModèle;
```

=

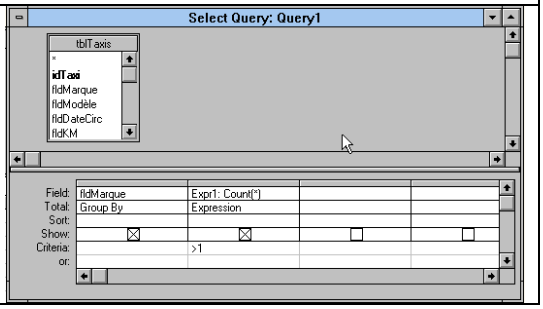



Afin d'implémenter la clause HAVING en Access-QBE , il suffit d'indiquer le critère de sélection des groupes dans la cellule du champ correspondant.

Affichez pour chaque marque, le nom de la marque et le nombre de taxis pour cette marque, en ignorant les marques pour lesquelles on dispose uniquement d'un seul taxi.

```
SELECT fldMarque, COUNT(*)
FROM tblTaxis
GROUP BY fldMarque
HAVING COUNT(*) > 1;
```

=



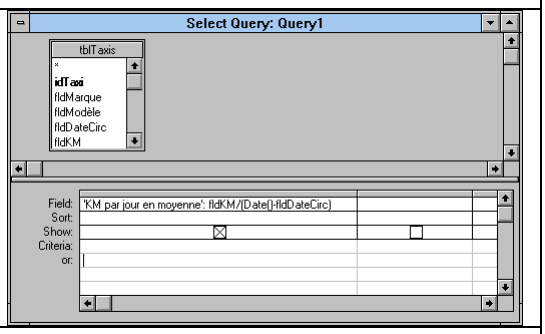
Champ à valeur calculée

Pour créer un champ à valeur calculée, vous devez indiquer manuellement l'expression dans la ligne intitulée **Field**: et préfixer l'expression par le nom à afficher suivi d'un double-point. La syntaxe est la même qu'en SQL.
Pour renommer un champ, il suffit de préfixer le nom du champ par le nom à afficher suivi d'un double-point.

Affichez pour chaque taxi, le numéro ainsi que le kilométrage parcouru en moyenne par jour.

```
SELECT idTaxi,fldKM/(Date()-
   fldDateCirc) AS 'KM par jour en
   moyenne'
FROM tblTaxis;
```

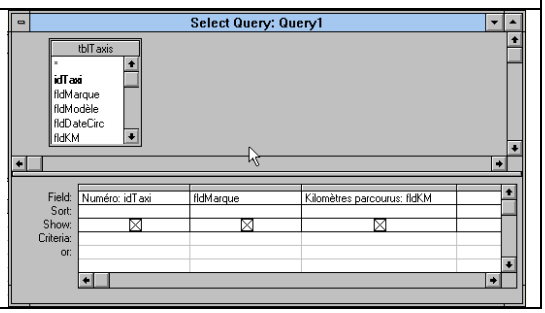
=




Affichez le numéro, la marque et le kilométrage de tous les taxis. L'en-tête du champ affichant le numéro doit porter le libellé "Numéro", celle du kilométrage doit porter le libellé "Kilomètres parcourus".

```
SELECT idTaxi AS Numéro, fldMarque,
   fldKM AS 'Kilomètres parcourus'
FROM tblTaxis;
```

=



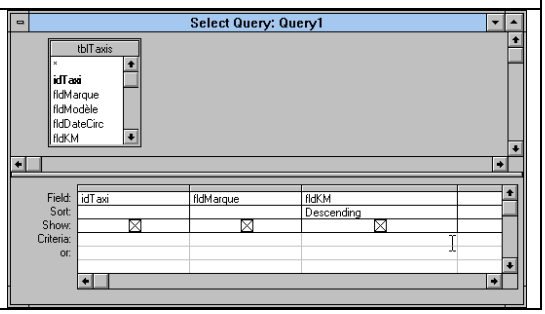

Trier une liste d'enregistrements

Access-QBE nous permet pour chaque champ sélectionné, de spécifier dans la ligne **Sort:** l'ordre du tri (ascendant/descendant).

Affichez une liste avec le numéro, la marque et le kilométrage parcouru de tous les taxis. Triez cette liste par ordre descendant sur le kilométrage.

```
SELECT idTaxi, fldMarque, fldKM
FROM tblTaxis
ORDER BY fldKM DESC;
```

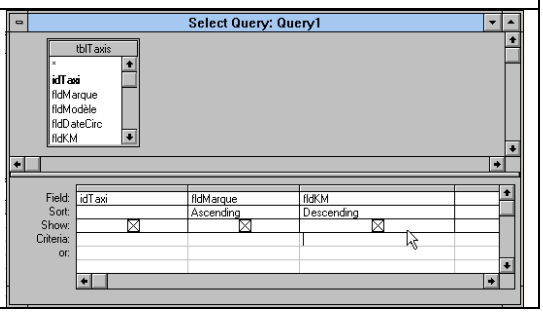
=



Même question. Triez cette fois par ordre alphabétique sur les marques et pour chaque marque par ordre descendant sur le kilométrage.

```
SELECT idTaxi, fldMarque, fldKM
FROM tblTaxis
ORDER BY fldMarque, fldKM DESC;
```

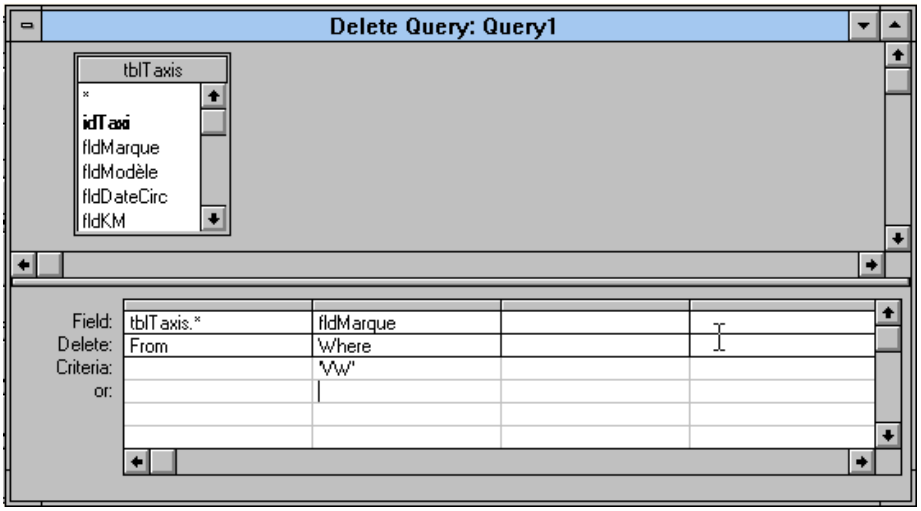
=




Requêtes de suppression

Voici à titre d'exemple un écran QBE qui représente une requête SQL, qui efface de la table *tblTaxis* tous les VW.

```
DELETE FROM tblTaxis
WHERE fldMarque='VW';
```




Une cellule appartenant à la ligne intitulée **Delete:** peut prendre en principe 2 valeurs possibles.

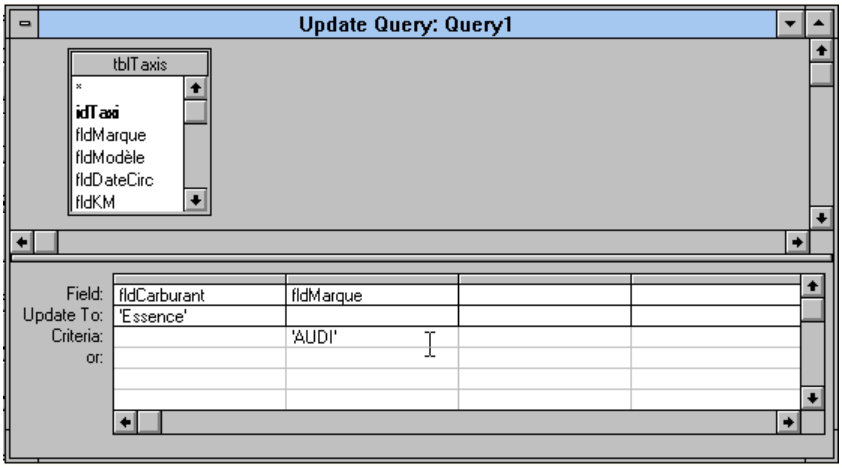
- **From** Dans ce cas, la cellule au dessus doit indiquer le nom de la table dans laquelle on veut effacer des enregistrements (Attention à la syntaxe <Nom de la table>.* )
- **Where** Permet de spécifier des critères de sélection

| ACTION | COMMENT FAIRE ? |
|---|---|
| Sélectionner le type de requête SUPPRESSION | |
| Exécuter la requête | |
| | |

Requêtes de modification

Voici à titre d'exemple un écran QBE qui représente une requête SQL, qui affecte la valeur 'Essence' au champ *fldCarburant* pour tous les Audi.

```
UPDATE tblTaxis
SET fldCarburant='Essence'
WHERE fldMarque='AUDI';
```




La ligne intitulée **Update To:** définit la nouvelle valeur pour le champ à modifier.

Access ne peut pas exécuter la requête de modification lorsque votre écran QBE contient l'astérisque (*) comme indicateur d'un champ sélectionné. Comme l'utilisation de l'astérisque est très utile pour les requêtes de sélection, il ne faut pas oublier de l'enlever après la conversion d'une requête de sélection en requête de modification.

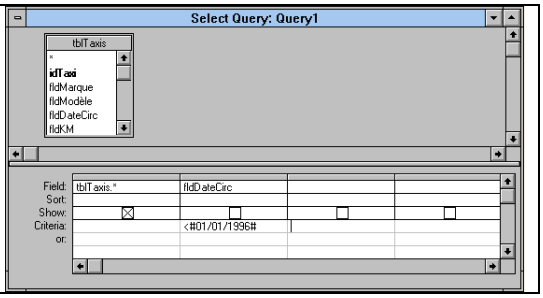
| ACTION | COMMENT FAIRE ? |
|---|---|
| Sélectionner le type de requête MODIFICATION | |
| Exécuter la requête | |
| | |

## THEORIE SUPPLEMENTAIRE - LES REQUETES PARAMETREES

Voici à titre d'exemple une requête, qui affiche toutes les informations concernant les taxis dont la date de première mise en circulation est antérieure au 1.1.96.

```
SELECT *
FROM tblTaxis
WHERE fldDateCirc<#1/1/96#;
```

=

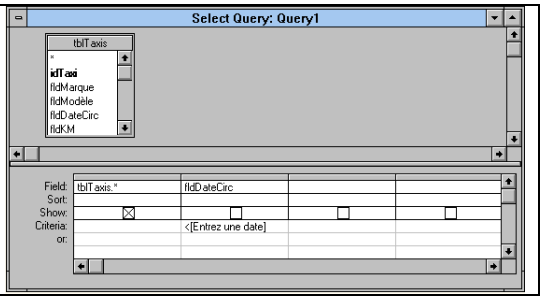



Imaginez que de temps en temps vous vouliez établir cette liste, mais avec une date de première mise en circulation différente. Au lieu de modifier à chaque fois le critère de sélection en SQL ou en QBE, vous définissez une requête paramétrée. Dans la requête paramétrée, vous n'indiquez pas directement la date (une constante), mais un paramètre (une variable), et à chaque fois que la requête est exécutée, une boîte de dialogue demande à l'utilisateur d'entrer une date avant l'exécution. Après que l'utilisateur a entré cette date, la requête affiche tous les taxis, dont la date de première mise en circulation est antérieure à la date entrée.

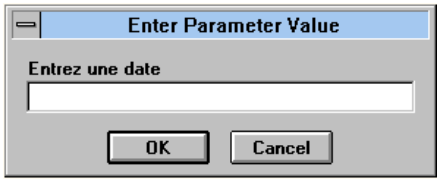
Voici un exemple SQL et QBE, qui vous montre qu'on indique simplement le texte à afficher dans la boîte de dialogue.

```
SELECT *
FROM tblTaxis
WHERE fldDateCirc<[Entrez une date];
```

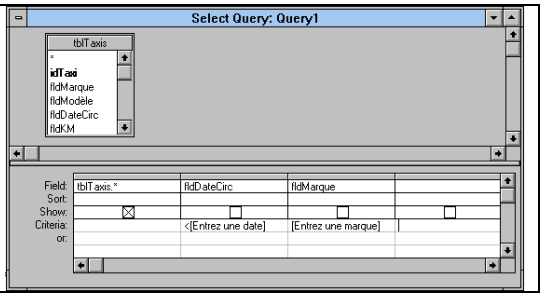
=



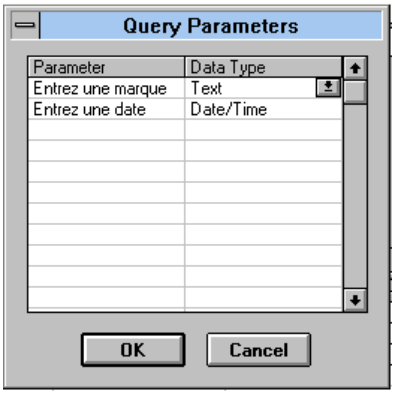

L'exécution de cette requête est précédée par l'affichage de la boîte de dialogue suivante.




Vous pouvez évidemment préciser plusieurs paramètres dans une requête. La requête suivante demande une marque ainsi que la date de la première mise en circulation.

| | | |
|---|---|---|
| `SELECT *`<br>`FROM tblTaxis`<br>`WHERE fldDateCirc<[Entrez une date]`<br>`AND fldMarque=[Entrez une marque];` | = |  |

Les boîtes de dialogue apparaissent dans l'ordre de définition des paramètres. Lorsque pour une raison ou l'autre, vous désirez modifier l'ordre d'apparence des boîtes de dialogue, sans modifier la requête, vous utilisez soit en SQL la commande **PARAMETERS** soit vous ouvrez en mode QBE la fenêtre **Query Parameters** et indiquez les paramètres dans l'ordre désiré.

| | | |
|---|---|---|
| `PARAMETERS [Entrez une marque] Text,`<br>`  [Entrez une date] DateTime;`<br>`SELECT *`<br>`FROM tblTaxis`<br>`WHERE fldDateCirc<[Entrez une date]`<br>`AND fldMarque=[Entrez une marque];` | = | Menu **Query/parameters**<br>  |

## ACTIVITES

1. Ouvrez la BD *taxis.mdb* (Vous allez créer différentes requêtes basées sur la table *tblTaxis*)

2. Créez via QBE une requête *qselACirc*, qui affiche le numéro , la marque et le modèle de tous les taxis mis en circulation après le 31.12.1995. Exécutez la requête. Analysez le code SQL que MS-Access a créé. Modifiez le code de façon à ce qu'il corresponde exactement à notre syntaxe. Réexécutez la requête. Qu'est ce que vous remarquez ?

Notez votre code SQL

3. Créez via SQL la requête *qselCarbKM*, qui affiche le numéro du taxi, la marque, le modèle ainsi que la date de première mise en circulation pour tous les taxis qui utilisent du carburant Diesel et qui ont un kilométrage entre 50.000 et 100.000 (les 2 valeurs incluses). Triez les enregistrements du résultat par ordre ascendant sur la marque. Après avoir exécuté la requête, affichez la requête en mode QBE.

4. Créez via QBE la requête *qselMarMod*, qui affiche toutes les informations des taxis dont le nom de marque commence par la lettre 'V' et dont le nom du modèle ne finit pas par la lettre 'T'. Notez le code SQL correspondant.

5. Créez via QBE une requête *qselKM*, qui permet à l'utilisateur de spécifier pour le kilométrage d'un taxi une limite inférieure ainsi qu'une limité supérieure, et qui affiche le numéro, la marque, le modèle et le kilométrage actuel de tous les taxis dont le kilométrage est une valeur qui se trouve entre ces limites.

   Notez le code SQL correspondant à cette spécification QBE.

6. A l'aide d'une requête *qinsMerc*, insérez l'enregistrement suivant dans la BD.

| | | | | | |
|---|---|---|---|---|---|
| 9 | Mercedes | 300SL | 15.7.1997 | 200 | |

   Pour l'instant nous laissons la valeur pour le carburant non définie. Utilisez pour ce champ la valeur NULL et non pas la chaîne vide "".

Quel mode est-ce que vous allez utiliser, QBE ou SQL ? Justifiez votre choix.

7. Créez via SQL une requête *qupdNull*, qui écrit la chaîne "Inconnu" dans le champ *fldCarburant* de chaque enregistrement ou ce champ est non défini. Exécutez la requête.

Affichez la requête en mode QBE.

8. Créez via QBE une requête *qselNouvVoit*, qui compte le nombre de voitures dont la date de première mise en circulation est antérieure au 1.1.1996. Notez le code SQL correspondant.

9. Utilisez la méthode QBE afin de résoudre les requêtes suivantes. Notez à chaque fois le code SQL correspondant.

| Enoncé | Code SQL |
|---|---|
| Affichez toutes les informations pour les taxis qui sont soit des VW , soit des Mercedes avec carburant Essence, soit des BMW avec moins de 10000 km. Triez les enregistrements par ordre ascendant sur les marques et par ordre descendant sur le kilométrage. | |

| | |
|---|---|
| Comptez le nombre de taxis qui sont actuellement plus 'vieux' que 2 années. (Nous supposons: 1 année = 365 jours) L'en-tête du champ affiché portera le titre 'Vieux taxis' | |
| Affichez les marques pour lesquelles on dispose exactement de X voitures. Le nombre X est à obtenir à chaque fois avant l'exécution de la requête, sans modifier la définition (SQL,QBE) de la requête. | |
| Affichez pour chaque marque, le nombre de voitures ainsi que la moyenne des kilomètres parcourus. | |
| Regroupez les taxis sur le mois de leur date de première mise en circulation, et affichez pour chaque groupe, le nombre de taxis. | |

# 11.4  TP No 4 : Les relations et les requêtes multitable

DUREE APPROX. : 4h.

**PRESENTATION**

Les requêtes en Access ne sont évidemment pas limitées à une seule table. De toute façon, Access supporte le langage SQL, ce qui nous offre déjà une possibilité de formuler nos requêtes multitable. Pendant ce TP, nous allons également examiner les possibilités offertes par Access-QBE en ce qui concerne les requêtes multitable.

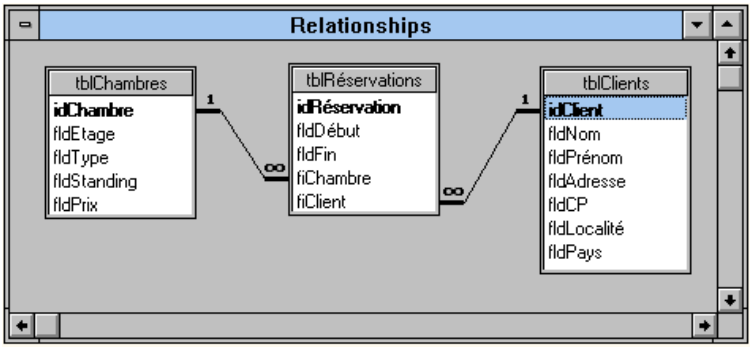
Comme le TP3, ce TP contient une partie théorique supplémentaire au cours. Cette partie nous fournit un aperçu sur les possibilités de réaliser des requêtes multitable à l'aide des relations définies entre les différentes tables.

**THEORIE SUPPLEMENTAIRE**

Les relations en Access

La partie théorique de ce cours nous a appris qu'une relation entre deux tables se fonde sur l'existence d'une clé étrangère (angl. Foreign Key) , qui fait référence à la clé primaire (ang. Primary Key) d'une autre table.

Voici, à titre d'exemple, trois tables utilisées par un hôtelier pour la gestion (simplifiée) de ses chambres.




**tblChambres**

| Nom du champ | Type de données | Description |
|---|---|---|
| **idChambre** | Numérique | Numéro de la chambre |
| fldEtage | Numérique | Numéro de l'étage |
| fldType | Texte [15] | Simple/Double |
| fldStanding | Texte [15] | Standard/Luxe/Grand Luxe |
| fldPrix | Numérique | Prix de la chambre par nuit (en Luf.) |

**tblRéservations**

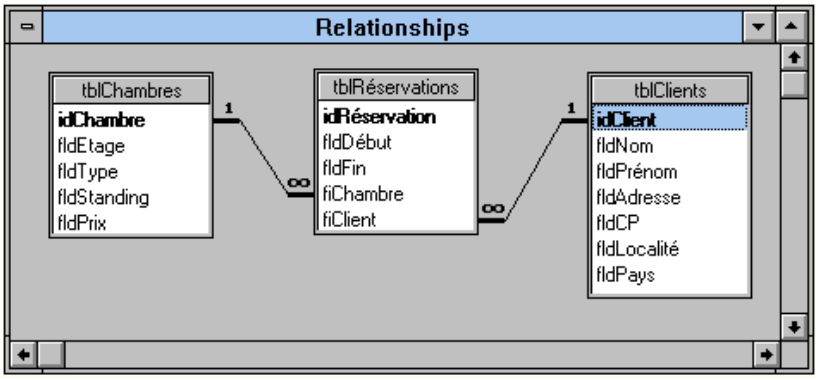
| Nom du champ | Type de données | Description |
|---|---|---|
| **idRéservation** | Numérique | Numéro de la réservation |
| fldDébut | Date | Début de la réservation |
| fldFin | Date | Fin de la réservation |
| fiChambre | Numérique | Numéro de la chambre |
| fiClient | Numérique | Numéro du client |

**tblClients**

| Nom du champ | Type de données | Description |
|---|---|---|
| **idClient** | Numérique | Numéro du client |
| fldNom | Texte [20] | |
| fldPrénom | Texte [20] | |
| fldAdresse | Texte [25] | Rue et numéro |
| fldCP | Texte [7] | |
| fldLocalité | Texte [20] | |
| fldPays | Texte [15] | |

Dans la table *tblRéservations*, les champs *fiChambre* et *fiClient* sont des clés étrangères, qui fournissent des références vers les clés primaires *idChambre* et *idClient* des tables *tblChambres* resp. *tblClients*. De cette façon, on peut pour chaque réservation déterminer avec exactitude le client et la chambre concernée.

Par le simple fait d'inclure des clés étrangères dans une table, Access ne reconnaît pas encore une relation entre les tables. Access ne sait par exemple pas automatiquement que le champ *fiChambre* de la table *tblRéservations* fait référence à la clé primaire *idChambre* de la table *tblChambres*. A l'aide de la fenêtre ***Relationships*** , on peut indiquer à Access les clés étrangères et leurs clés primaires associées, et ainsi définir les relations entre les tables.




La documentation de Access nous dit que le fait d'établir des relations simples entre des tables, nous facilite en outre le création de requêtes multitable. Nous allons vérifier cette affirmation plus tard dans la partie **Activités** de ce TP.

Un avantage incontestable de la fenêtre ***Relationships*** est la possibilité de définir ce qu'on appelle **l'intégrité référentielle** pour une relation donnée.

L'intégrité référentielle

 Par **contrainte d'intégrité référentielle** (angl. Referential Integrity Constraint) , on entend l'obligation qu'à chaque valeur de la clé étrangère corresponde une et une seule valeur de la clé primaire associée. Cette obligation doit toujours être vérifiée lors de l'ajout, de la suppression ou de la modification de données.

Il existe plusieurs scénarios qui peuvent compromettre l'intégrité référentielle d'une BD. Donnez à chaque fois un exemple précis en vous basant sur nos 3 tables *tblRéservations*, *tblChambres* et *tblClients*.

A. L'ajout d'une clé étrangère pour laquelle il n'existe pas de valeur correspondante dans la clé primaire associée.

   Exemple:

B. La modification d'une clé étrangère vers une valeur pour laquelle il n'existe pas de valeur correspondante dans la clé primaire associée.

   Exemple:

C. La suppression d'une clé primaire qui est référencée par une ou plusieurs valeurs d'une clé étrangère.

   Exemple:

D. La modification d'une clé primaire qui est référencée par une ou plusieurs valeurs d'une clé étrangère.

   Exemple:

Access nous offre plusieurs possibilités pour garantir à tout moment l'intégrité référentielle des données d'une BD.

I. **Interdiction** des opérations du type A, B, C et D. Bien que cette possibilité soit très efficace, il existe parfois une alternative préférable en fonction de la nature des données.

| | |
|---|---|
| II. | **Cascade** des opérations du type C et D vers les clés étrangères correspondantes. Une modification d'une clé primaire aurait comme conséquence la modification de toutes les clés étrangères correspondantes. Une suppression d'une clé primaire par exemple aurait comme conséquence la suppression automatique de tous les enregistrements dont la clé étrangère a la même valeur. Cette option est à utiliser avec précaution !!! |

**MANIPULATIONS IMPORTANTES**

Les relations

| **ACTION** | **COMMENT FAIRE ?** |
|---|---|
| Ouvrir la fenêtre ***Relationships*** | |
| Ajouter une table dans la fenêtre Relationships | |
| Effacer une table de la fenêtre Relationships | |
| Définir une relation entre deux tables | |
| Effacer une relation existante | |
| Modifier une relation existante | |
| Forcer le respect de l'intégrité référentielle | |
| Activer les suppressions resp. modifications en cascade | |
| | |
| | |

Les requêtes multitable

| **ACTION** | **COMMENT FAIRE ?** |
|---|---|
| Sélectionner les tables impliquées dans une requête multitable | |
| Réaliser une auto-jointure sur une table | |
| Afficher les noms des tables dans la partie inférieure de la fenêtre QBE (Utile pour les auto-jointures) | |

| | |
|---|---|
| • Sélectionner une requête existante comme base de la nouvelle requête | |
| • | |
| • | |

**ACTIVITES**

1. Créez une nouvelle BD, que vous appelez *Hotel.mdb* .

2. Créez les 3 tables *tblChambres*, *tblRéservations* et *tblClients*.

3. Entrez les données suivantes dans les 3 tables.

**Table: tblChambres**

| idChambre | fldEtage | fldType | fldStanding | fldPrix |
|---|---|---|---|---|
| 1 | 1 | Double | Luxe | 4000 |
| 2 | 1 | Double | Grand Luxe | 4800 |
| 3 | 1 | Double | Luxe | 4000 |
| 4 | 1 | Double | Standard | 2800 |
| 5 | 1 | Double | Standard | 2800 |
| 6 | 1 | Double | Standard | 2800 |
| 7 | 2 | Simple | Standard | 3200 |
| 8 | 2 | Simple | Standard | 3200 |
| 9 | 2 | Simple | Luxe | 4400 |
| 10 | 2 | Double | Luxe | 3800 |

Record: 1 of 10

**Table: tblRéservations**

| idRéservation | fldDébut | fldFin | fiChambre | fiClient |
|---|---|---|---|---|
| 1 | 03.08.1998 | 07.08.1998 | 4 | 2 |
| 2 | 04.08.1998 | 07.08.1998 | 6 | 3 |
| 3 | 20.08.1998 | 01.09.1998 | 4 | 1 |
| 4 | 01.09.1998 | 02.09.1998 | 1 | 2 |

Record: 1 of 4

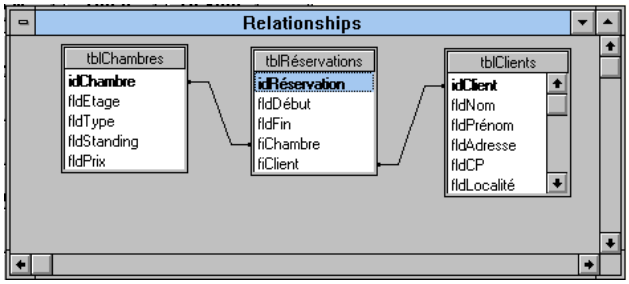
Attention: Le format des dates peut être différent sur votre PC (Entrez p.ex. simplement 3 8 98 lors de l'entrée d'une date)

**Table: tblClients**

| idClient | fldNom | fldPrénom | fldAdresse | fldCP | fldLocalité | fldPays |
|---|---|---|---|---|---|---|
| 1 | Weber | Jos | 23, rue Principale | L-6678 | Grevenmacher | Luxembourg |
| 2 | Hinterwald | Franz-Josef | 4, Hinter der Hecke | D-23987 | Oberwaldheim | Deutschland |
| 3 | Baguette | Colette | 28, av des Boulangers | F-56743 | Four-les-Croissants | France |

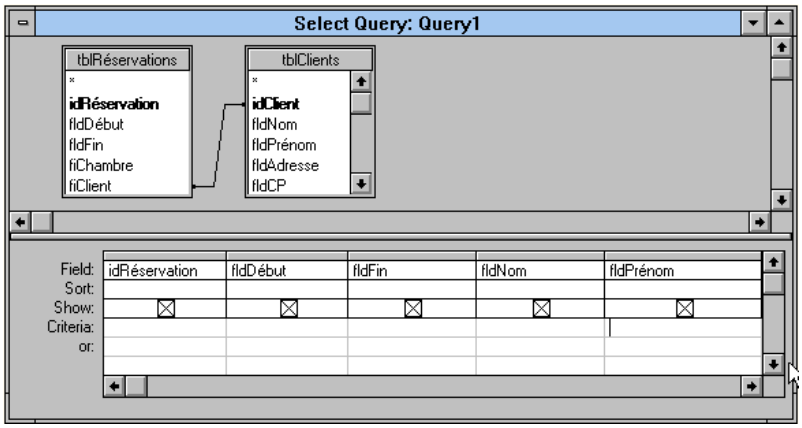
Record: 3 of 3

4. Créez des simples relations (sans intégrité référentielle) entre les tables *tblRéservations* et *tblChambres* ainsi que entre les tables *tblRéservations* et *tblClients*.




5. Quel est le code SQL d'une requête qui affiche pour chaque réservation, le numéro de réservation, les dates de début et de fin de réservation, ainsi que le nom et le prénom du client concerné.

Créez cette requête via QBE et sauvegardez la sous le nom *qselResCli*




Remarquez que la relation entre les deux tables *tblRéservations* et *tblClients* est automatiquement affichée dans la fenêtre QBE.

Exécutez la requête pour voir le résultat.

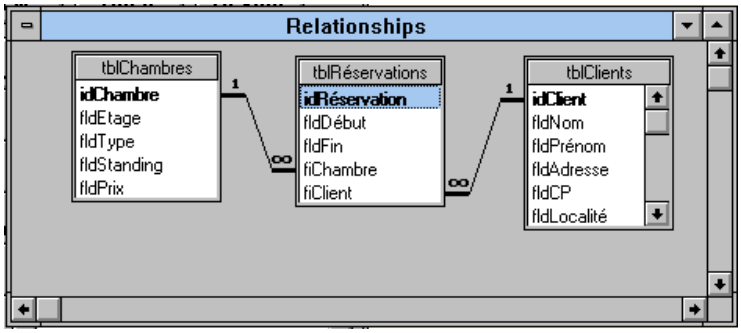
Notez le code SQL que Access vient de créer.

A part des différences syntaxiques que nous connaissons déjà du TP3, qu'est-ce que vous remarquez lorsque vous comparez votre code SQL à celui créé par Access ?

Dans la fenêtre ***Relationships*** , effacez la relation entre *tblRéservations* et *tblClients*, et ouvrez par après le requête *qselResCli* en mode QBE. Qu'est ce que vous remarquez ?

Effacez la relation entre les deux tables *tblRéservation* et *tblClients* également dans la fenêtre QBE. Vérifiez maintenant encore une fois le code SQL géré par Access et exécutez encore une fois la requête. Quel est donc l'utilité d'une relation simple en MS-Access ?

6. Utilisez la fenêtre ***Relationships*** afin de créer entre les tables *tblRéservation* et *tblChambres* ainsi que entre les tables *tblRéservations* et *tblClients* des relations avec intégrité référentielle. Utilisez à chaque fois l'option **One To Many** et ne sélectionnez pas les cases à cocher concernant la modification resp. l'effacement en cascade.




Effacez dans la table *tblClients* le client numéro 2. Expliquez la réaction de MS-Access.

Pour le client numéro 3 , essayez d'affecter la valeur 4 au champ *idClient*. Expliquez la réaction de MS-Access.

Insérez dans la table *tblRéservations* la réservation suivante. Expliquez la réaction de MS-Access.

| idRéservation | fldDébut | fldFin | fiChambre | fiClient |
|---|---|---|---|---|
| 5 | 03.09.98 | 10.09.98 | 9 | 5 |

Pour la réservation numéro 2 , essayez d'affecter la valeur 15 au champ *fiChambre*. Expliquez la réaction de MS-Access.

7. Utilisez la fenêtre ***Relationships*** afin de modifier les relations entre les tables *tblRéservations* et *tblChambres* ainsi que entre les tables *tblRéservations* et *tblClients*. Sélectionnez maintenant pour chaque relation, les 2 cases à cocher concernant la modification resp. l'effacement en cascade.

Effacez dans la table *tblClients* le client numéro 2. Expliquez la réaction de MS-Access.

Pour le client numéro 3 , essayez d'affecter la valeur 4 au champ *idClient*. Expliquez la réaction de MS-Access.

Insérez dans la table *tblRéservations* la réservation suivante. Expliquez la réaction de MS-Access.

| idRéservation | fldDébut | fldFin | fiChambre | fiClient |
|---|---|---|---|---|
| 5 | 03.09.98 | 10.09.98 | 9 | 5 |

Pour la réservation numéro 2 , essayez d'affecter la valeur 15 au champ *fiChambre*. Expliquez la réaction de MS-Access.

8. Créez via SQL la requête *qselCliEt1* qui affiche le nom, le prénom , l'adresse, le code postal, la localité et le pays des clients ayant réservé une chambre à la première étage pendant le mois d'août 1998.

Affichez l'écran QBE pour cette requête. Qu'est ce que vous remarquez ?

9. Créez et exécutez via SQL la requête suivante. On cherche les noms et prénoms de tous les clients ayant déjà une fois réservé la même chambre que le client numéro 4. Sauvegardez la requête sous le nom *qselCli4*.

Affichez et analysez l'écran QBE correspondant à cette requête SQL. Essayez ensuite de refaire la requête en utilisant uniquement les possibilités QBE. Sauvegardez cette requête sous le nom *qselCli4bis*.

10. Soit la requête suivante

```
SELECT fldNom, fldPrénom
FROM tblClients
WHERE idClient IN (SELECT fiClient
                   FROM tblRéservations
                   WHERE fiChambre=6);
```

A première vue, cette requête est difficile à implémenter en Access-QBE. Trouvez un moyen pour représenter cette requête en Access-QBE. Expliquez votre méthode.

11. Utilisez si possible la méthode QBE afin de résoudre les requêtes suivantes. Notez à chaque fois le code SQL tel qu'il est créé par MS-Access.

| Enoncé | Code SQL généré par Access |
|---|---|
| Affichez une liste triée par ordre ascendant des numéros des chambres dont la période de réservation contient le 1.9.98. | |
| Affichez le numéro de chambre, les dates de début et de fin de réservation ainsi que le nom et le prénom du client pour toutes les réservations qui commencent après la date actuelle et dont le client vient du Luxembourg. | |
| Affichez le numéro de réservation, le nom et le prénom du client pour toutes les réservations de la même chambre que celle utilisée par le client 4 entre le 4.8.98 et le 7.8.98. (Auto-jointure) | |

| | |
|---|---|
| Affichez pour chaque chambre la recette totale de toutes les réservations. | |
| Affichez une liste avec les numéros et les prix des chambres n'ayant pas du tout été réservées pendant les mois de juillet et d'août 1998. | |

# 11.5  TP No 5 : Les formulaires (Création à l'aide d'un assistant et utilisation)

DUREE APPROX. : 2h.

**PRESENTATION**

MS-Access nous offre des assistants (angl. wizards) afin de nous alléger considérablement le travail d'élaboration des différents composants d'une BD. Access contient plusieurs types d'assistants:

- Assistant de table
- Assistant de requête
- Assistant de formulaire
- Assistant de rapport

Ces assistants existent en plus en différentes versions, en fonction du composant qu'on veut créer. Il existe par exemple pour les formulaires un "Assistant de Formulaire Colonne Simple", un "Assistant de Formulaire Tabulaire", un "Assistant de Formulaire Graphique", mais également par exemple un "Assistant de Bouton de commande", qui est à notre disposition sur n'importe quel type de formulaire.

Beaucoup de tâches ayant atteint un certain degré de complexité peuvent donc être facilitées par l'utilisation d'un assistant.

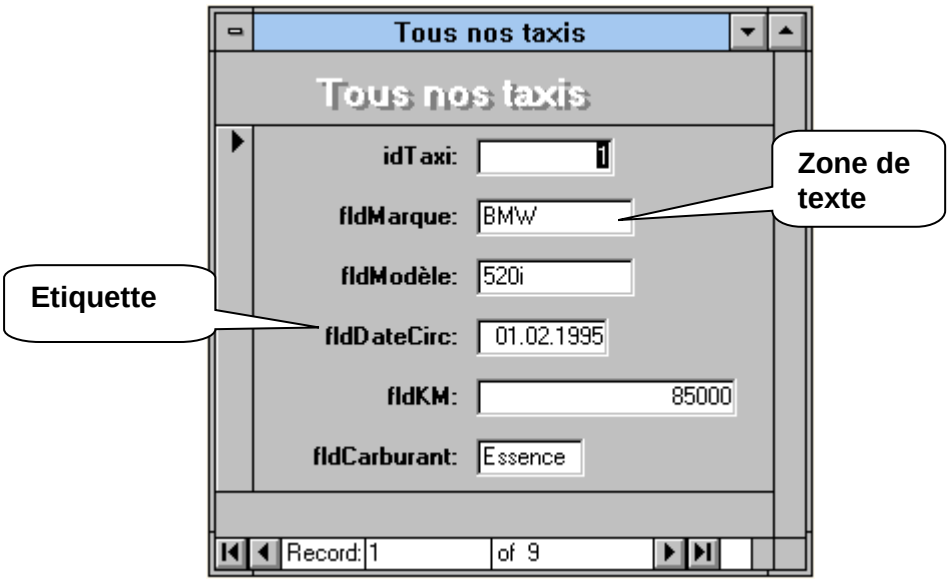
Que fait l'assistant ?

Un assistant pose un certain nombre de questions en relation avec le composant qu'on veut créer. Basé sur les réponses, l'assistant crée le composant pour nous.

Pour cet exercice nous allons nous baser sur la table *tbltaxis* (*taxis.mdb*), créée lors du **TP2**..

| **Nom du champ** | **Type de données** | **Description** |
|---|---|---|
| idTaxi | Compteur | Numéro identificateur d'un taxi |
| fldMarque | Texte [15] | Marque du taxi |
| fldModèle | Texte [15] | Modèle du taxi |
| fldDateCirc | Date | Date de la première mise en circulation |
| fldKm | Numérique | Kilométrage actuel |
| fldCarburant | Texte [10] | Essence/Diesel |

Nous utilisons l'"Assistant de Formulaire Colonne Simple" (Single-Column Wizard), afin de créer le formulaire suivant.




Ce formulaire contient des contrôles, comme par exemple les zones de texte ou les étiquettes. L'apparence ainsi que le comportement de ces contrôles sont déterminés par leurs propriétés. Une zone de texte possède par exemple une propriété *Back Color*, qui détermine la couleur d'arrière-plan de la zone. Nous allons analyser les propriétés plus en détail lors du prochain TP, pour l'instant, retenez uniquement la notion.

Modes

Tout comme pour les autres composants d'une BD, l'utilisateur a le choix parmi plusieurs modes d'affichage pour n'importe quel formulaire. Nous avons en principe pour chaque composant un mode "structure" (=mode "création") et un mode "utilisation" (=mode "Feuille de Données"). En ce qui concerne les formulaires, Access nous offre un mode en plus ce qui nous donne les 3 modes suivants:

1. Le mode "*Création"*
   → Définir des nouveaux formulaires ou modifier des formulaires existants. Placer les contrôles sur le formulaire et définir les propriétés des contrôles et du formulaire.

2. Le mode "*Formulaire"*
   → Entrer, consulter et modifier les données. Ce mode affiche un formulaire comme il a été défini en mode création.

3. Le mode "Feuille de Données"
   → Entrer consulter et modifier les données. Ce mode affiche une "Feuille de Données" similaire à celles qui représentent directement les données des tables et des requêtes.

**MANIPULATIONS IMPORTANTES**

Création d'un formulaire à l'aide d'un assistant

| **ACTION** | **COMMENT FAIRE ?** |
|---|---|
| Créer un nouveau formulaire en utilisant un assistant | |

| | |
|---|---|
| Avancer d'une étape dans l'exécution d'un assistant | |
| Reculer d'une étape dans l'exécution d'un assistant | |
| Finir l'exécution d'un assistant (Attention: Finir trop tôt ne mène pas au résultat attendu) | |
| | |

Remarque:

La création d'un formulaire à l'aide d'un assistant nous évite de créer un formulaire en mode création. Un formulaire créé par un assistant peut bien sûr être utilisé en mode formulaire, ou mode Feuille de Données, et peut même être modifié en mode création.

| **ACTION** | **COMMENT FAIRE?** |
|---|---|
| Sélectionner le mode Création | |
| Sélectionner le mode Formulaire | |
| Sélectionner le mode Feuille de Données | |
| | |

Utilisation du formulaire

Nous utiliserons en général le mode Formulaire ou le mode Feuille de Données pour entrer, consulter et modifier des données.

| **ACTION** | **COMMENT FAIRE?** |
|---|---|
| Sélectionner l'enregistrement suivant (seulement formulaire Colonne Simple et Tabulaire) | |
| Sélectionner l'enregistrement précédent (seulement formulaire Colonne Simple et Tabulaire) | |
| Sélectionner un enregistrement quelconque (seulement formulaire Colonne Simple et Tabulaire) | |
| Sélectionner le premier enregistrement (seulement formulaire Colonne Simple et Tabulaire) | |
| Sélectionner le dernier enregistrement (seulement formulaire Colonne Simple et Tabulaire) | |
| Sélectionner un champ quelconque | |
| Passer au prochain champ | |

| | |
|---|---|
| Reculer au champ précédent | |
| Annuler la dernière action | |
| Editer le contenu d'un champ | |
| !!! Access sauvegarde automatiquement chaque modification lorsque vous quittez l'enregistrement modifié. Comment est-ce qu'on peut forcer la sauvegarde d'un enregistrement à tout moment ? | |
| Ajouter un nouvel enregistrement | |
| | |

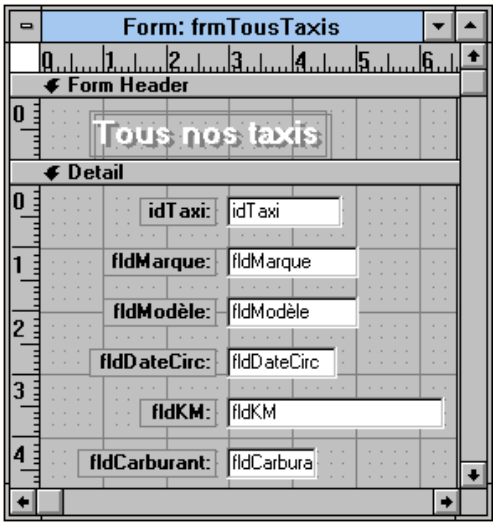
**ACTIVITES**

1. Ouvrez la BD *taxis.mdb*

2. Créez un formulaire du type "Colonne Simple", basé sur la table *tblTaxis* en utilisant l'assistant correspondant. Voici les indications nécessaires pour répondre aux questions de l'assistant:
   - Tous les champs de la table apparaissent dans leur ordre original.
   - Utilisez le style "Embossed".
   - Le titre du formulaire sera "Tous nos taxis".
   - Le formulaire sera tout de suite ouvert en mode "Formulaire" (Affichage des données).

3. Sauvegardez le nouveau formulaire sous le nom *frmTousTaxis*.

4. Affichez le formulaire *frmTousTaxis* en mode "Création".




5. Affichez le formulaire en mode "Feuille de Données".

Tous nos taxis

| idTaxi: | fldMarque: | fldModèle: | fldDateCirc: | fldKM: | fldCarburant: |
|---|---|---|---|---|---|
| 1 | BMW | 520i | 01.02.1995 | 85000 | Essence |
| 2 | Ford | Scorpio | 15.07.1994 | 110800 | Diesel |
| 3 | VW | Passat | 28.09.1996 | 56700 | Diesel |
| 4 | BMW | 525i | 10.10.1996 | 48500 | Essence |
| 5 | Audi | A6 | 09.11.1997 | 4500 | Diesel |
| 6 | VW | GolfTDI | 23.09.1997 | 7800 | Diesel |
| 7 | VW | Passat | 22.02.1996 | 59000 | Diesel |
| 8 | VW | Passat | 13.10.1994 | 100800 | Diesel |
| 9 | Mercedes | 300SL | 15.07.1997 | 200 | Inconnu |
| (Counter) | | | | 0 | |

Record: 1 of 9

6. Affichez le formulaire en mode "Formulaire".


7. En mode "Formulaire", cherchez manuellement l'enregistrement pour lequel le carburant indique toujours "Inconnu" et remplacez cette valeur par "Essence"

8. Ajoutez un nouvel enregistrement vierge (Notez que la valeur *idTaxi* de type compteur est automatiquement générée).

9. Entrez les valeurs suivantes pour les champs

| | |
|---|---|
| fldMarque | Opel |
| fldModele | Omega |
| fldDateCirc | 10.08.1997 |
| fldKM | 190 |
| fldCarburant | Essence |

10. Comment est-ce que vous pouvez sauvegarder ce nouvel enregistrement ?

11. Affichez le formulaire en mode "Feuille de Données" et élargissez la fenêtre de façon à ce qu'elle affiche tous les champs.

12. Sachant que le mode "Feuille de Données" d'un formulaire correspond au mode "Feuille de Données" d'une table, triez les enregistrements en ordre croissant sur la marque.

13. Sélectionnez le mode "Formulaire". Est-ce que les enregistrements sont encore triés ? Si oui, ordonnez les enregistrements de nouveau dans leur ordre original.

14. Imprimez le formulaire en mode Formulaire. Que constatez-vous ?

---

15. Définissez un filtre qui affiche uniquement les taxis mis en circulation après le 1.1.1996, et appliquez ce filtre. Est-ce que ce filtre a un effet dans les deux modes (mode "Formulaire" et mode "Feuille de données") ?

16. Effacez le filtre (= réaffichez tous les enregistrements).

17. Fermez le formulaire.

# 11.6  TP No 6 : Les formulaires (Création sans assistant)

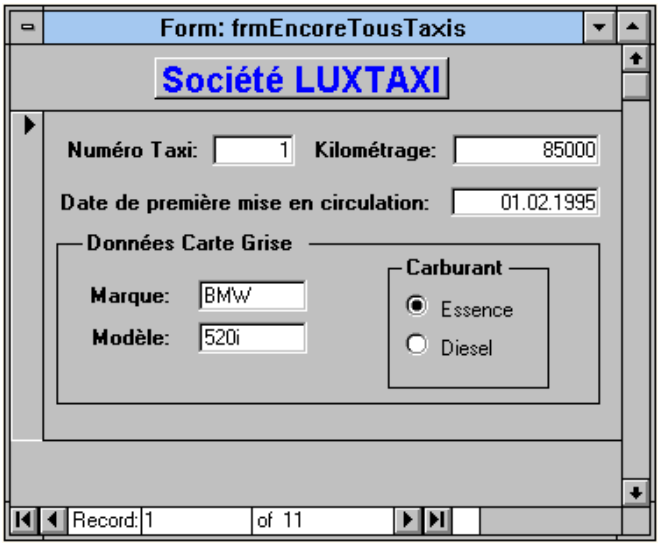
DUREE APPROX : 3h.

**PRESENTATION**

Ce TP nous apprendra à manipuler un contrôle sur un formulaire à l'aide de ses propriétés. MS-Access supporte la philosophie des objets, que vous avez déjà rencontrée en classe de 12CG, lors de l'étude détaillée de MS-Excel. En Excel, les objets importants étaient les classeurs (angl. workbook), les feuilles de calculs (angl. worksheet), et les champs (angl. range). Par analogie, chaque BD créée à l'aide de MS-Access, est une collection d'objets, tels que les tables, requêtes, formulaires etc. , qui sont eux-mêmes de nouveau composés d'autres objets comme par exemple les zones de liste, cases à cocher, zones de texte ou boutons d'options d'un formulaire. Les contrôles sur un formulaire sont donc également des objets, avec des propriétés, nous permettant de modifier l'apparence et le comportement des contrôles.

Quels sont les inconvénients du formulaire que nous avons créé à l'aide d'un assistant lors du dernier TP ?

Dans ce TP nous allons créer le formulaire *frmEncoreTousTaxis* sans utiliser un assistant. Ce formulaire affichera les mêmes données que le formulaire *frmTousTaxis*, mais l'apparence du formulaire sera un peu améliorée selon la façon suivante.



Par rapport au formulaire créé par l'assistant, nous avons changé la disposition des contrôles, nous avons un regroupement de certaines données techniques figurant sur la carte grise et nous avons choisi la représentation par groupe de boutons d'options en ce qui concerne le carburant d'un taxi. A chaque champ est associé une étiquette, qui indique à l'utilisateur d'une façon claire et nette la signification du champ.

Quel est l'avantage d'utiliser un groupe avec deux boutons d'options pour le carburant ?

Nous allons commencer par un nouveau formulaire vierge et ajouter un après l'autre les contrôles.

Le formulaire ainsi que chaque contrôle possèdent des propriétés qui sont affichées dans une petite fenêtre appelée **fenêtre des propriétés**. Cette fenêtre affiche à tout moment les propriétés du formulaire , respectivement du contrôle actuellement sélectionné. Nous pouvons changer l'apparence ainsi que le comportement du formulaire et des contrôles en modifiant quelques propriétés. Pour avoir une description détaillée d'une propriété d'un objet, utilisez le système d'aide en ligne de MS-Access.

Chaque formulaire peut contenir plusieurs sections. Il existe plusieurs types de sections dans un formulaire. Utilisez votre documentation Access et donnez pour chaque section d'un formulaire une petite définition:

| | |
|---|---|
| **En-tête d'un formulaire** **(angl. Form header)** | |
| **En-tête de page** **(angl. Page header)** | |
| **Section détail** **(angl. Detail section)** | |
| **Pied de page** **(angl. Page footer)** | |
| **Pied de formulaire** **(angl. Form footer)** | |

**MANIPULATIONS IMPORTANTES**

Création d'un formulaire sans un assistant (resp. modification en mode Création):

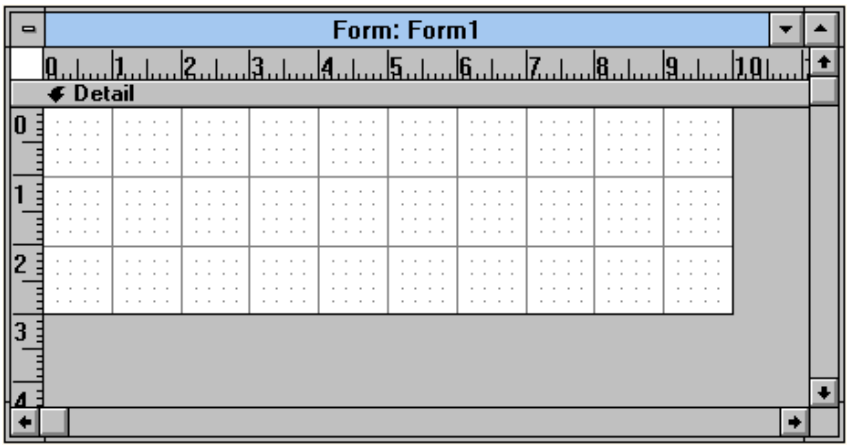
| **ACTION** | **COMMENT FAIRE ?** |
|---|---|
| Créer un nouveau formulaire vierge sans utiliser un assistant | |
| Afficher la fenêtre des propriétés | |
| Afficher la boite d'outils (angl. Toolbox) | |
| Afficher un en-tête resp. un pied de formulaire | |

| | |
|---|---|
| Afficher une en-tête resp. un pied de page | |
| Sélectionner un contrôle | |
| Sélectionner plusieurs contrôles | |
| Placer respectivement bouger le contrôle sur le formulaire | |
| Afficher les propriétés pour un contrôle ou pour le formulaire dans la fenêtre des propriétés | |
| Modifier une propriété spécifique d'un contrôle ou du formulaire | |
| Modifier la taille d'un contrôle avec la souris | |
| Dupliquer un contrôle | |
| Aligner plusieurs contrôles | |
| Adapter la largeur (angl. size) de plusieurs contrôles | |
| Egaliser l'espacement horizontal (angl. horizontal spacing) entre plusieurs contrôles | |
| Egaliser l'espacement vertical (angl. vertical spacing ) entre plusieurs contrôles | |
| Utilisation/Non-utilisation par défaut de l'assistant pour les groupes d'options, les zones de listes … | |
| Indiquer resp. modifier la suite du parcours des champs lorsque l'utilisateur utilise la touche TAB | |
| Indiquer resp. modifier le nom d'un contrôle | |
| Sélectionner uniquement l'étiquette d'un contrôle | |
| Sélectionner uniquement le contrôle sans l'étiquette | |
| Sélectionner le contrôle avec l'étiquette | |
| | |
| | |

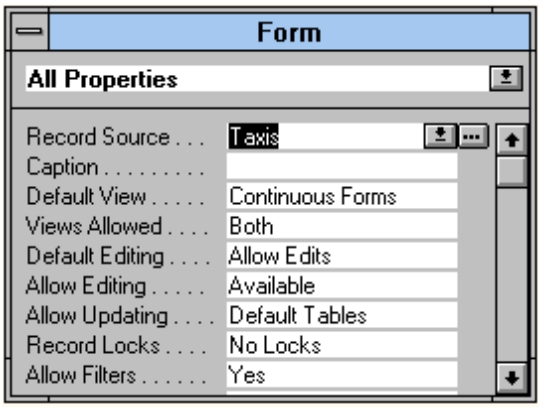
## ACTIVITES

1. Ouvrez la BD *taxis.mdb.*

2. Créez un nouveau formulaire qui est basé sur la table *tblTaxis*, sans utiliser un assistant. Un nouveau formulaire vierge apparaît en mode création.




3. Affichez la fenêtre des propriétés et la boite d'outils, qui contient tous les contrôles à votre disposition.

| **Fenêtre des propriétés** | **Boite d'outils** |
|---|---|
|  |  |





Dans la fenêtre des propriétés, affichez **toutes** les propriétés. Pour avoir de l'aide concernant une propriété, il suffit de positionner le curseur sur la propriété dans la fenêtre des propriétés, et d'utiliser la touche **F1**.

4. Choisissez comme couleur d'arrière plan (angl. *Back Color*) de la section détail du formulaire la couleur gris clair.

5. Affichez l'en-tête resp. le pied du formulaire.

6. Au milieu de l'en-tête du formulaire, placez un contrôle du type étiquette. Cette étiquette contiendra le texte "Société LUXTAXI" formaté en Arial 15, caractères gras, coloré en bleu. L'arrière plan de l'étiquette sera coloré en gris clair (comme l'arrière plan du formulaire) et comme effet supplémentaire utilisez l'option "élevé" (angl. raised). Manipulez les propriétés suivantes pour réaliser cette apparence.

| **Propriété** | **Valeur** |
|---|---|
| *Font Name* | |
| *Font Size* | |
| *Font Weight* | |
| *Fore Color* | |
| *Back Color* | |
| *Special Effect* | |

7. Alignez le texte au milieu de l'étiquette. Alignez l'étiquette au milieu de l'en-tête.

8. Placez une zone de texte pour le numéro du taxis sur le formulaire. Vous voyez qu'une zone de texte possède une étiquette attachée à la zone.

   Sélectionnez uniquement cette étiquette et sélectionnez comme couleur d'arrière-plan la couleur gris clair (comme celle du formulaire). Entrez comme texte (prop. *Caption*) "**Numéro Taxi** :", et formatez ce texte en caractère gras. Alignez le texte à droite de l'étiquette.

   Sélectionnez maintenant la zone de texte et choisissez comme source (prop. *Control Source*) le champ *idTaxi* de la table *tblTaxis*. Sélectionnez comme effet supplémentaire l'option "abaissé" (angl. sunken).

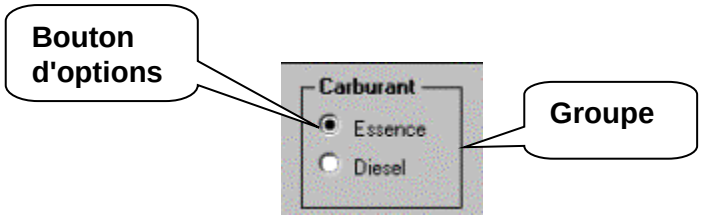
   Afin que le formulaire affiche uniquement un enregistrement à la fois, sélectionnez la propriété *Default View* du formulaire, et choisissez la valeur "Single Form".

   Sélectionnez le mode "Formulaire" pour vérifier l'aspect du formulaire.

9. Sauvegardez le formulaire sous le nom *frmEncoreTousTaxi* pour être sûr de ne pas perdre les fruits de votre travail.

10. Créez maintenant de façon analogue les autres contrôles du formulaire.

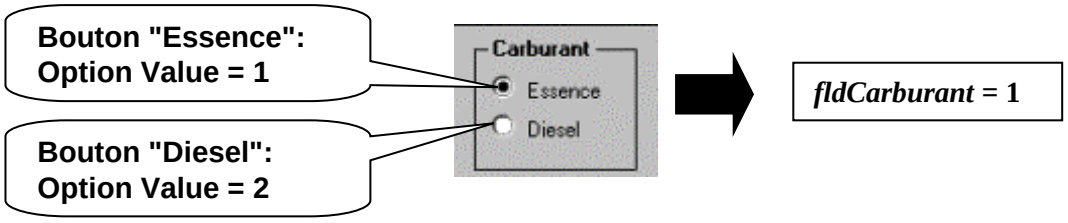
    Pour les boutons d'option créez d'abord le groupe.



En ne vous servant pas de l'assistant, attachez le groupe au champ *fldCarburant*. Créez les 2 boutons d'options et placez les à l'intérieur du groupe. Sélectionnez la valeur *sunken* pour la propriété *Special Effect* des deux boutons. L'étiquette du premier bouton doit afficher "Essence", celle du deuxième bouton "Diesel".

**ATTENTION:**

Nous venons de toucher à une des limites de MS-Access. La représentation par boutons d'options du champ carburant implique que la valeur stockée dans le champ *fldCarburant* soit une valeur numérique. Access ne sait pas stocker le texte correspondant à un bouton d'options directement dans la table, mais uniquement la valeur numérique de la propriété *Option Value*, du bouton actuellement sélectionné.




Ceci implique un travail supplémentaire pour les enregistrements existants. Nous devons convertir les valeurs du champ *fldCarburant* de façon que la valeur "Essence" est convertie en valeur "1" et la valeur "Diesel" est convertie en valeur "2". Heureusement, nous n'avons pas besoin de modifier le type de donnée du champ *fldCarburant*. Les valeurs 1 et 2 peuvent être stockées en tant que valeurs textes.

Cette conversion peut être réalisée à l'aide de deux requêtes de modification, dont le code SQL est:

_____

_____

_____

Vérifiez ensuite que la propriété *Option Value* du bouton "Essence" vaut 1 et celle du bouton "Diesel" vaut 2. Sélectionnez le mode "Formulaire" et vérifiez que le type de carburant est correctement affiché pour les enregistrements existants.

En ce qui concerne les nouveaux enregistrements, nous voulons que le groupe d'options propose comme carburant par défaut "Essence". Il suffit de sélectionner la propriété valeur par défaut (angl. *Default Value*) du groupe d'options et d'entrer la valeur 1.

11. Utilisez votre formulaire pour ajouter l'enregistrement suivant:

| | | | | | |
|---|---|---|---|---|---|
| **(Valeur générée)** | **Audi** | **A6** | **10.7.97** | **3000** | **Diesel** |

12. Adaptez votre formulaire de façon à ce que les champs soient parcourus dans l'ordre suivant

    1. Marque
    2. Modèle
    3. Carburant
    4. Kilométrage
    5. Date de première mise en circulation

    Le champ Numéro taxi n'est pas accessible.

13. Supposons que la société de taxis envisage uniquement l'achat et l'utilisation de taxis des marques BMW, Audi, VW, Ford, Mercedes, Opel, Renault et Citroën. Pour ces marques les modèles sont bien sûr connus, mais on doit tenir compte du fait que de nouveaux modèles apparaissent sur le marché.
    Comment est-ce qu'on pourrait encore améliorer le formulaire afin de respecter au maximum possible le principe de la saisie minimale pour l'utilisateur.

    ---

    ---

    Effectuez ces modifications sur votre formulaire.

14. Sauvegardez le formulaire

# 11.7 TP No 7 : Les formulaires basés sur plusieurs tables

DUREE APPROX : 4h.

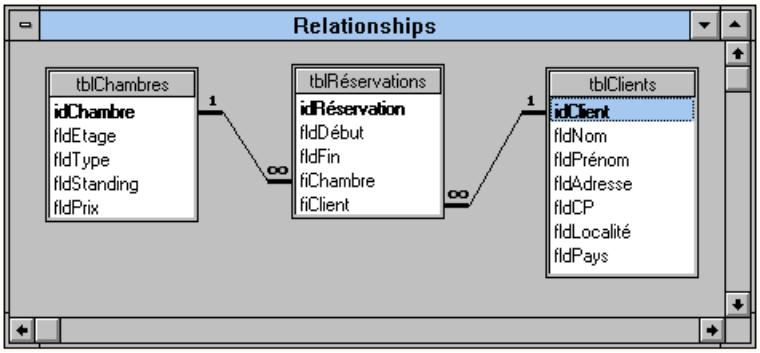
**PRESENTATION**

Sachant que les tables d'une BD sont généralement liées à l'aide de relations, les formulaires nous offrent la possibilité d'afficher les données de plusieurs tables dans un seul formulaire.

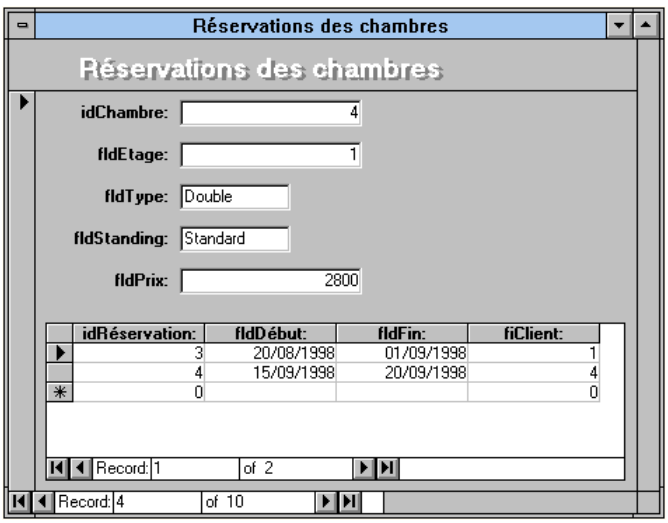
Les données provenant de deux tables, qui sont reliées de façon 1:N à l'aide d'une clé étrangère, qui fait référence à une clé primaire, peuvent par exemple être très bien représentées à l'aide d'un formulaire principal/secondaire (angl. Main/Subform).

Exemple:

Soient les trois tables suivantes, créées lors du **TP4** (*Hotel.mdb*).




Le formulaire suivant représente des informations en provenance des deux tables *tblChambres* et *tblRéservations*. En occurrence, le formulaire est capable d'afficher pour chaque chambre, les réservations correspondantes.




**Formulaire principal**

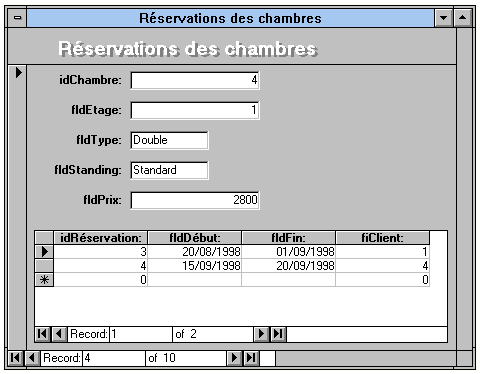
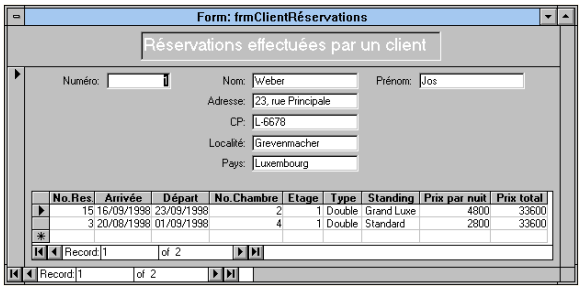
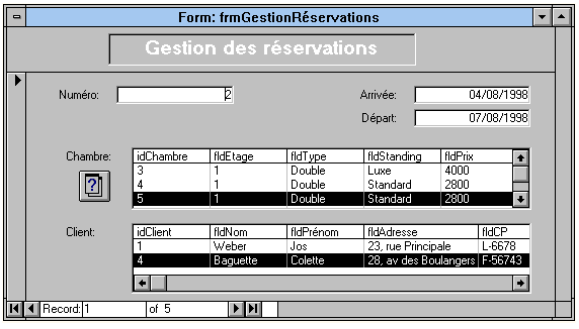
**1:N**

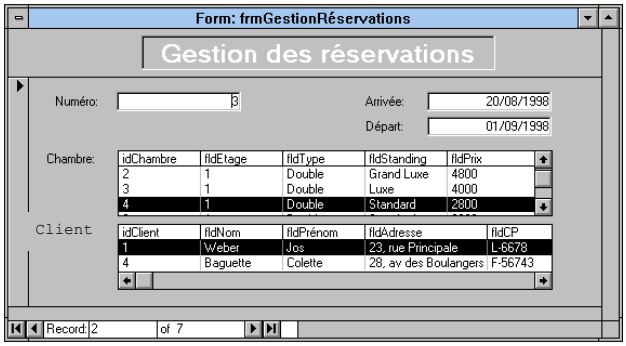
**Formulaire secondaire**

Le formulaire secondaire (ou sous-formulaire) affiche toutes les réservations effectuées pour la chambre numéro 4. Au moment où on sélectionne une chambre différente dans le formulaire principal, les réservations de cette chambre seront affichées.

Le rôle d'un formulaire secondaire est d'afficher tous les enregistrements; dont la valeur de la clé étrangère (généralement non affichée) correspond à la valeur le la clé primaire de l'enregistrement actuel du formulaire principal.

Ce TP est réparti sur quatre parties A, B,C et D en vue de son envergure, avec chaque fois un formulaire à créer resp. à modifier.

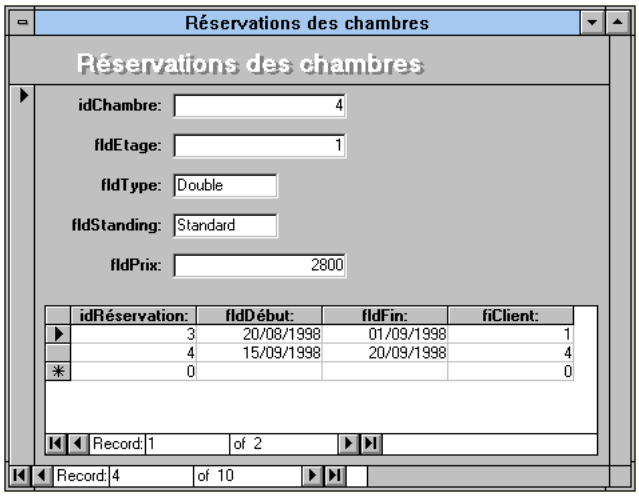
| Partie | Formulaire créé |
|---|---|
| **A** | *frmChambreRéservations*<br>  |
| **B** | *frmClientRéservations*<br>  |
| **C** | *frmGestionRéservations*<br>  |

| | |
|---|---|
| **D** | *frmGestionRéservations* version améliorée  |



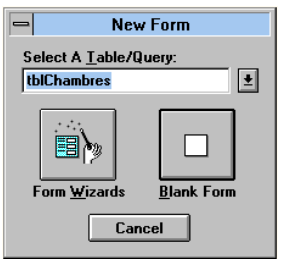
**ACTIVITES**

PARTIE A : CREATION DU FORMULAIRE *frmChambreRéservations* A L'AIDE D'UN ASSISTANT

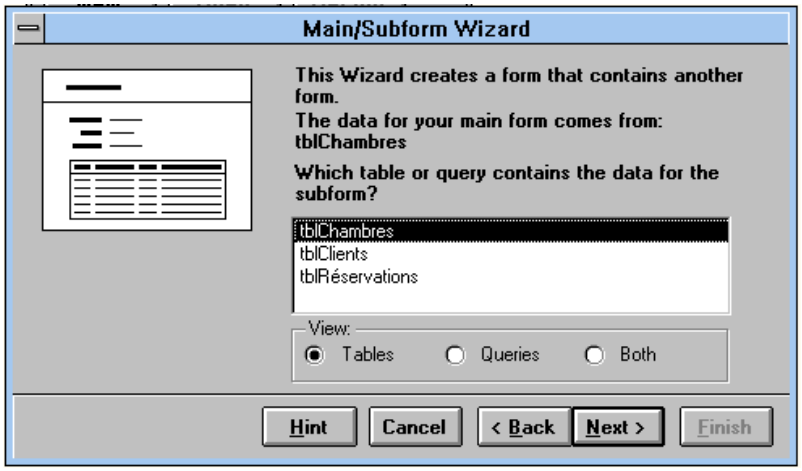



1. Ouvrez la BD *Hotel.mdb* .

2. Nous allons créer un formulaire, qui est capable d'afficher pour chaque chambre les réservations effectuées. Pour cela, nous utilisons l'"Assistant de Formulaire principal/secondaire" (angl. Main/Subform Wizard). Avant d'exécuter l'assistant, MS-Access vous demande le nom de "la table" concernée, ne sachant pas encore que deux tables différentes sont à la base de notre formulaire. Indiquez dans cette boîte de dialogue le nom de la table sur laquelle sera basée le formulaire principal, donc *tblChambres*.



3. Après avoir exécuté l'assistant, la fenêtre suivante vous est proposée.




L'assistant vous indique que vous êtes en train de créer un formulaire principal/secondaire et que *tblChambres* est à la base du formulaire principal. Vous devez maintenant sélectionner la table sur laquelle se base le sous-formulaire, donc *tblRéservations*.

4. Les deux prochaines boîtes de dialogue vous permettent de sélectionner pour chaque table resp. pour chaque formulaire (principal et secondaire) les champs à afficher.

   En ce qui concerne le formulaire principal (*tblChambres*), sélectionnez tous les champs disponibles.

   Pour le formulaire secondaire (*tblRéservations*), sélectionnez tous les champs à l'exception de la clé étrangère *fiChambre*.

   Pourquoi est-ce qu'on n'affiche en principe pas dans le sous-formulaire la clé étrangère, qui fait le lien avec l'enregistrement actuel du formulaire principal ?

5. Voici les indications nécessaires pour répondre aux prochaines questions de l'assistant:
   - Utilisez le style "Embossed".
   - Le titre du formulaire sera "Réservations des chambres".

   Avant de créer le formulaire, l'assistant vous indique que vous devez d'abord sauvegarder le sous-formulaire[1].

[1] Ceci est dû à la philosophie des objets supportée par MS-Access. En fait, le sous-formulaire est un formulaire indépendant, qui est simplement imbriqué comme objet dans le formulaire principal.

**Convention des noms:**

Les noms des sous-formulaires sont précédés du préfixe **fsub**.

Sauvegardez le sous-formulaire sous le nom *fsubChambreRéservations*.

MS-Access vous affiche sur le formulaire pour chaque chambre automatiquement les réservations correspondantes.

6. Sauvegardez le formulaire principal sous le nom *frmChambreRéservations.*

7. Affichez le sous-formulaire *fsubChambreRéservations* de façon indépendante du formulaire principal. Quels enregistrements et quels champs sont affichés ?

8. Affichez le formulaire *frmChambreRéservations* en mode création. Affichez ensuite les propriétés du sous-formulaire, et expliquez brièvement le rôle des propriétés suivantes du sous-formulaire.

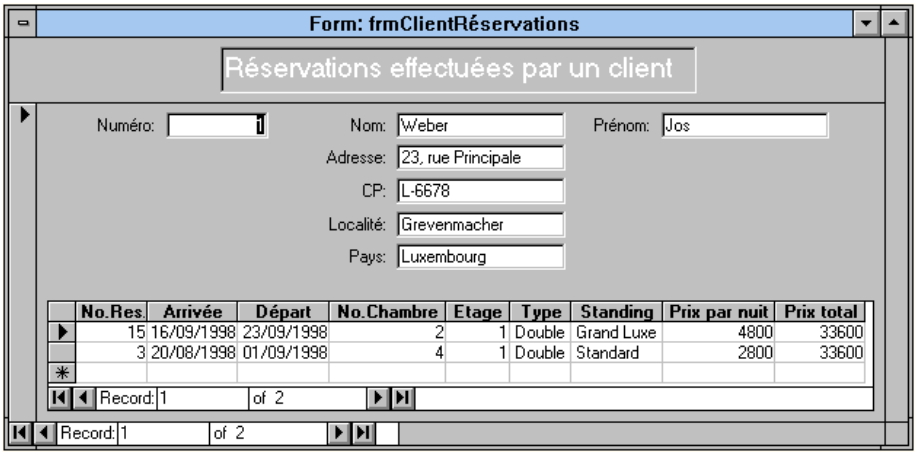
| **Propriété** | **Description** |
|---|---|
| Source Object | |
| Link Child Fields | |
| Link Master Fields | |

9. Utilisez le formulaire *frmChambreRéservations* en mode formulaire pour sélectionner la chambre numéro 2. Ajouter à l'aide du formulaire une nouvelle réservation, dont le numéro est 15, la date d'arrivée le 16 septembre 1998 et la date de départ le 23 septembre 1998. Le client concerné est le client numéro 1.

   Affichez ensuite le contenu de la table *tblRéservations*. Quelle est la valeur du champ fiChambre _____ pour le nouvel enregistrement ?

   Quel est le désavantage de ce formulaire en relation avec l'ajout de nouvelles réservations ?

## PARTIE B : CREATION DU FORMULAIRE *frmClientRéservations* SANS ASSISTANT




Il est important de comprendre; qu'un formulaire principal ainsi qu'un formulaire secondaire; sont basés chacun sur un objet différent. Dans l'exemple précédent, cet objet était une table. Cependant, il s'avère parfois nécessaire de baser un formulaire sur une requête[1], comme dans le cas de cet exemple, où on ne veut pas simplement voir le numéro d'une chambre concernée par une réservation dans le sous-formulaire, mais également l'étage, le type, le standing, le prix par nuit etc. Dans ce cas, le sous-formulaire n'est plus basé sur la table *tblRéservations*, mais sur une requête, qui affiche pour chaque réservation, les données de la réservation et celles de la chambre correspondante (jointure). En plus, nous allons calculer le prix total de chaque réservation via un champ à valeur calculée. Ce formulaire sera créé sans assistant.

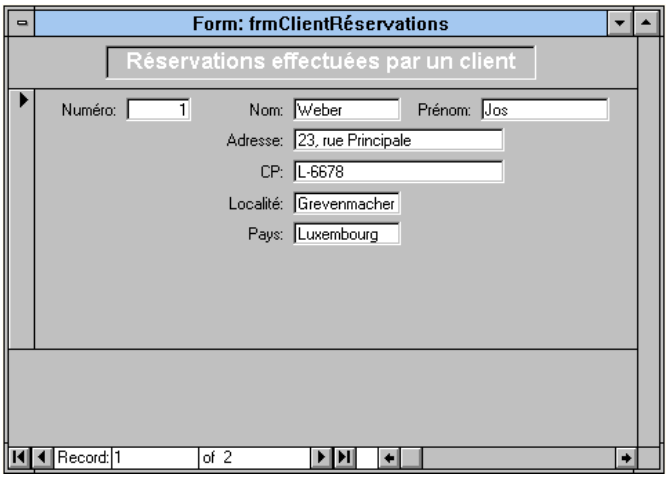
**Etapes de création du formulaire:**

- Créer le formulaire principal.
- Définir la requête pour le sous-formulaire.
- Créer le sous-formulaire basé sur la requête.
- Intégrer le sous-formulaire dans le formulaire principal.

1. Nous commençons par le formulaire principal, qui se base sur la table _____ .

   Créez ce formulaire du type colonne simple sans utiliser un assistant. Sauvegardez le formulaire sous le nom *frmClientRéservations* .

[1] Règle générale: Lorsque des données représentées sur un formulaire proviennent de plusieurs tables, il faut utiliser une requête comme base du formulaire. Il est impératif de comprendre que l'on parle des données spécifiées par la propriété 'Control Source' du formulaire seulement, et non pas de celles représentées dans une zone de liste attachée à une table ou requête.




2. Créez une requête *qselRéservationChambre* , qui affiche pour chaque réservation les champs suivants.

   - Le numéro de réservation.
   - La date de début de la réservation.
   - La date de fin de la réservation.
   - Le numéro de la chambre correspondante.
   - Le numéro de l'étage.
   - Le type de la chambre.
   - Le standing de la chambre.
   - Le prix par nuit de la chambre.
   - Le prix total de la réservation (Champ calculé).
   - Le numéro du client.

3. Sans utiliser l'assistant, créez un nouveau formulaire *fsubClientRéservations*, qui affiche tous les champs de la requête *qselRéservationChambre*, à l'exception du numéro client.

   Ne vous souciez pas trop de la disposition des zones de texte, puisque nous allons plus tard intégrer ce formulaire comme sous-formulaire dans *frmClientRéservations* et l'afficher en mode feuille de données. Indiquez les étiquettes suivantes pour les différents champs.

   | | |
   |---|---|
   | Le numéro de réservation | No. Rés. |
   | La date de début de la réservation | Arrivée |
   | La date de fin de la réservation | Départ |
   | Le numéro de la chambre correspondante | No. Chambre |
   | Le numéro de l'étage | Etage |
   | Le type de la chambre | Type |
   | Le standing de la chambre | Standing |
   | Le prix par nuit de la chambre | Prix par nuit |
   | Le prix total de la réservation | Prix total |

   Affectez ensuite la valeur *Datasheet* à la propriété *Default View* du formulaire. Ceci fait, le formulaire est toujours affiché en mode feuille de données.

4. Ouvrez le formulaire *frmClientRéservations* en mode création, et ajoutez un contrôle pour un sous-formulaire (Icône  de la boîte d'outils).

   Modifiez les propriétés du sous-formulaire de façon à ce qu'il affiche le formulaire *fsubClientRéservations* avec uniquement les réservations qui ont été effectuées par le client actuel du formulaire principal. Indiquez les propriétés ainsi que les valeurs que vous avez affectées à celles-ci.

| **Propriété** | **Description** |
|---|---|
| | |
| | |
| | |
| | |

   Effacez l'étiquette du sous-formulaire.

5. Affichez le formulaire *frmClientRéservations* en mode formulaire. Quel est le désavantage de ce formulaire en ce qui concerne l'ajout et la modification de nouvelles réservations pour un client donné ?

   _____

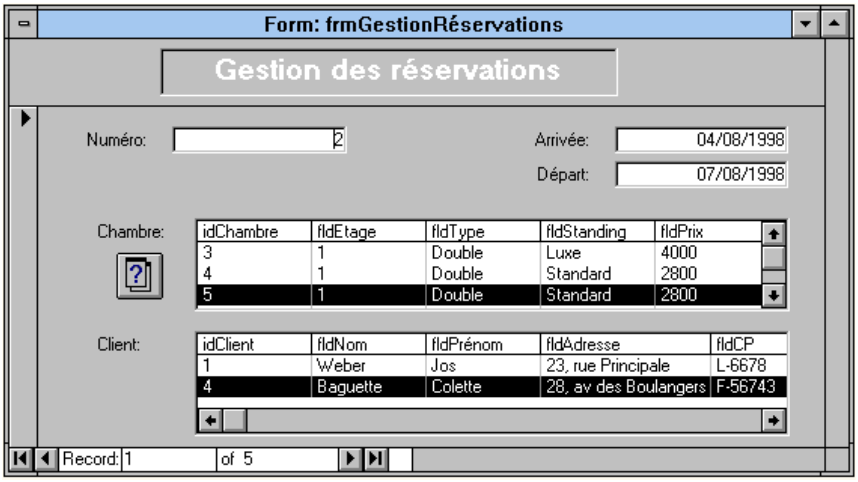
   _____

6. Nous allons utiliser ce formulaire uniquement pour la consultation des réservations par client. A cette fin, nous allons bloquer toute possibilité d'ajout ou de modification de données.

   Pour cela, affectez la valeur *Read Only* à la propriété *Default Editing* du formulaire. Essayez maintenant en mode formulaire d'ajouter ou de modifier des enregistrements. Qu'est-ce que vous remarquez ?

   _____

## PARTIE C : CREATION DU FORMULAIRE *frmGestionRéservations* SANS ASSISTANT




Ce formulaire, nous permettant de consulter, d'ajouter et de modifier des réservations, devra nous présenter les avantages suivants:

- Pour une nouvelle réservation, on peut uniquement sélectionner une chambre, qui existe effectivement dans le système. L'utilisateur sélectionne simplement la chambre dans la zone de liste, et le numéro de cette chambre sera automatiquement repris dans l'enregistrement correspondant de la table *tblRéservations*. Ceci évite des erreurs de frappe lors de l'entrée manuelle des informations concernant une chambre.

- Même remarque pour les clients.

- La sélection du bouton de commande fait apparaître le formulaire *frmChambreRéservations* (voir Partie A), qui affiche pour la chambre actuellement saisie, toutes les réservations correspondantes. De cette façon, l'utilisateur du formulaire peut s'assurer que la chambre qu'il veut affecter à une nouvelle réservation n'est pas déjà occupée pendant la période correspondante.

**Etapes de création et d'adaptation du formulaire:**

- Créer le formulaire avec les différents contrôles, dont deux zones de liste se basant chacune sur une table différente.
- Ajouter un bouton à l'aide duquel l'utilisateur peut afficher le formulaire *frmChambreRéservations*.

1. Créez le nouveau formulaire *frmGestionRéservations*, qui se base sur la table *tblRéservations*, sans utiliser l'assistant. Ajoutez ensuite l'en-tête avec le titre ainsi que les champs concernant le numéro de réservation ainsi que la date d'arrivée et de départ.

2. Ajoutez une zone de liste pour afficher les chambres en désactivant l'assistant des zones de liste. Cette zone de liste doit en fait afficher tous les enregistrements de la table _____ . Lorsque l'utilisateur sélectionne un enregistrement (une chambre), la valeur

du champ *idChambre* doit automatiquement être transférée dans le champ _____ de la table *tblRéservations*.

Pour les réservations existantes, la chambre affectée à une réservation est automatiquement sélectionnée dans la zone de liste. Afin de réaliser ce comportement, vous devez vous servir des propriétés suivantes de la zone de liste (Utilisez le système d'aide en ligne pour avoir des informations sur les différentes propriétés).

| **Propriété** | **Valeur** |
|---|---|
| Control Source | |
| Row Source Type | |
| Row Source | |
| Column Count | |
| Column Widths | |
| Bound Column | |
| Column Heads | |

Nommez la zone de liste *lstChambres*, et n'oubliez pas de sauvegarder votre formulaire de temps en temps !

3. Créer la zone de liste pour les clients simplement en faisant une copie de la zone de liste des chambres. Modifiez ensuite les propriétés nécessaires pour que cette liste affiche tous les enregistrements de la table _____ . Lorsque l'utilisateur sélectionne un enregistrement (un client), la valeur du champ *idClient* doit automatiquement être transférée dans le champ _____ de la table *tblRéservations*.

| **Propriété** | **Valeur** |
|---|---|
| Control Source | |
| Row Source Type | |
| Row Source | |
| Column Count | |
| Column Widths | |
| Bound Column | |
| Column Heads | |

4. Affichez votre formulaire en mode formulaire et testez son fonctionnement par exemple en ajoutant une nouvelle réservation. Vous allez remarquer que la manipulation des réservations est assez confortable, et que les erreurs de frappe sont minimisées.

   Toutefois, le formulaire présente encore le défaut qu'on risque d'affecter une chambre à une réservation, tandis que cette chambre est déjà occupée pendant un ou plusieurs jours de la réservation.

Au début de ce TP, nous avons créé le formulaire *frmChambreRéservations*, qui nous affiche pour une chambre donnée toutes les réservations. Nous allons définir un bouton de commande, afin d'afficher automatiquement ce formulaire sur besoin.

5. En mode création, activez l'assistant des contrôles, et créez le bouton de commande, qui se trouve à gauche de la zone de liste avec les chambres.

   Voici quelques indications importantes pour répondre aux questions de l'assistant.

   - Le bouton doit implémenter l'opération **Ouvrir** sur le **formulaire frmChambreRéservations.**
   - Le formulaire doit afficher tous les enregistrements (pour l'instant).
   - Affichez **l'image CueCards** sur le bouton.
   - Acceptez le nom que MS-Access propose pour le bouton de commande.

   Testez le fonctionnement du bouton en mode formulaire. Comment est-ce qu'on pourrait encore améliorer le fonctionnement dans le sens de faciliter le travail de l'utilisateur ?

---

6. Pour ce faire, nous devons nous occuper un peu du concept des **événements**. Chaque contrôle connaît un certain nombre d'événements prédéfinis tel que le fait de cliquer sur un bouton ou de modifier la valeur d'un champ d'un formulaire. On a la possibilité d'associer des procédures contenant du code Access Basic (VBA en Access97) à un tel événement.

   A titre d'exemple, lorsqu'on crée une procédure en AccessBasic, qui contient la commande `MsgBox("Bonjour!")`, et qu'on associe cette procédure à l'événement *On Click* d'un bouton de commande, une boite de dialogue avec le texte Bonjour sera affichée à chaque fois que l'utilisateur clique sur le bouton, c.à.d. à chaque fois que se passe événement *On Click* pour le bouton en question.

   En mode création, sélectionnez la propriété *On Click* du bouton. Cette propriété est en fait liée à un événement.

   En cliquant sur le bouton  vous pouvez voir le code actuel de cette procédure. Ce code a été créé automatiquement par l'assistant.

   La partie intéressante du code est la suivante

```
Dim DocName As String
Dim LinkCriteria As String

DocName="frmChambreRéservations"
DoCmd OpenForm DocName , , , LinkCriteria
```

   Les trois premières lignes devraient constituer du "Déjà-vu" lorsque vous pensez à la programmation de MS-Excel. On définit les deux variables **`DocName`** et **`LinkCriteria`**, toutes les deux du type String. Elles peuvent donc contenir du texte.

La commande **DoCmd OpenForm DocName , , , LinkCriteria** ouvre le formulaire dont le nom est indiqué dans la variable **DocName** (donc *frmChambreRéservations*). A l'aide de la valeur spécifiée dans **LinkCriteria**, on peut définir quel enregistrement sera l'enregistrement actuel lorsque le formulaire *frmChambreRéservations* est ouvert.

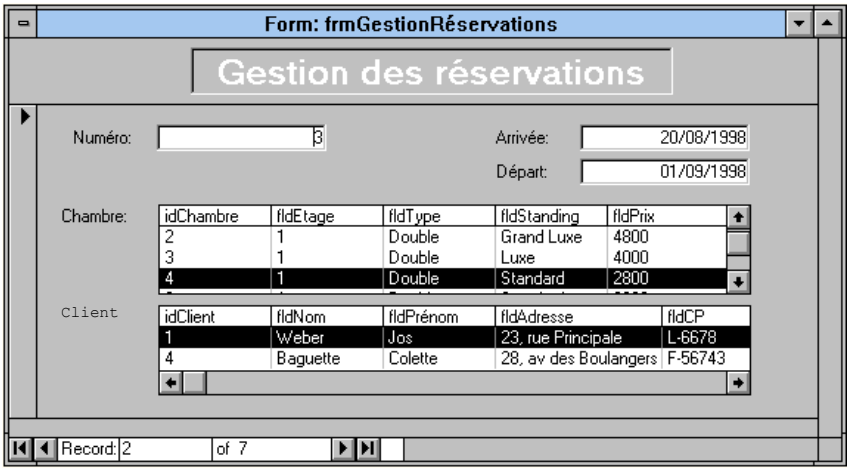
La chambre affichée dans le formulaire *frmChambreRéservations* devra correspondre à la chambre actuellement sélectionnée dans la zone de liste correspondante. La valeur de *idChambre* dans *frmChambreRéservations* devra donc correspondre à la valeur actuelle de *idChambre* dans le zone de liste dans *frmGestionRéservations*.

A ces fins, ajoutez la ligne suivante avant la commande **DoCmd OpenForm ...**

```
LinkCriteria="[idChambre]=Forms!frmGestionRéservations!lstChambres"
```

Sauvegardez le formulaire et testez le fonctionnement de votre formulaire en mode formulaire.

PARTIE D : AMELIORATION DE LA FONCTIONNALITE DU FORMULAIRE *frmGestionRéservations*




Vous voyez effectivement encore le même formulaire, que celui de la partie précédente, avec la seule exception qu'on n'a plus besoin du bouton de commande, parce que la zone de liste avec les chambres ne contient dès le départ que les chambres qui ne sont pas encore occupées dans la période entre la date d'arrivée et la date de départ.

**Etapes:**

- Effacer le bouton de commande créé lors de la partie précédente.
- Créer une requête paramétrée (voir TP3), qui sera à la base de la zone de liste avec les chambres.
- Attacher la requête paramétrée à la zone de liste.
- Prévoir une réexécution automatique de la requête dans certains cas.

1. Ouvrez le formulaire *frmGestionRéservations* en mode création et effacez le bouton de commande.

2. Créez via SQL la requête paramétrée *qselChambresLibres*, qui affiche tous les champs des chambres non occupées pendant la période indiquée par les dates d'arrivée et de départ du formulaire *frmGestionRéservations*. Utilisez pour l'instant les deux paramètres **[Entrez la date d'arrivée]** et **[Entrez la date de départ]**.

Testez le fonctionnement de la requête !

3. En ce qui concerne la zone de liste avec les chambres, vous devez changer la propriété *Row Source* à la valeur *qselChambresLibres*. Ceci fait, la zone affiche en principe les enregistrements de cette requête. Testez votre formulaire en mode formulaire. Qu'est-ce que vous remarquez ?

---

La requête *qselChambresLibres* attend deux valeurs entrées comme paramètres avant de s'exécuter. Comme ces deux valeurs se trouvent déjà sur le formulaire *frmGestionRéservations* (date d'arrivée, date de départ), on n'a pas envie de les spécifier encore une fois dans les boîtes de dialogue. Il existe un moyen pour dire à MS-Access que les valeurs des paramètres proviennent directement d'un ou de plusieurs contrôles du formulaire *frmGestionRéservations*.

Renommez sur le formulaire le champ qui contient la date d'arrivée en *txtArrivée*, et celui qui contient la date de départ en *txtDépart*.

Ouvrez ensuite la requête en mode SQL et remplacez les deux paramètres par la syntaxe suivante

**[Entrez la date d'arrivée]** → `[Forms]![frmGestionRéservations]![txtArrivée]`
et
**[Entrez la date de départ]** → `[Forms]![frmGestionRéservations]![txtDépart]`

4. Testez ensuite le formulaire. Vous remarquez que la zone de liste affiche en principe uniquement les chambres non occupées pendant le période de la date d'arrivée à la date de départ spécifiée dans le formulaire. Toutefois, il existe encore deux petits problèmes à résoudre.

   En général, la requête *qselChambresLibres* est exécutée une seule fois lors de l'ouverture du formulaire. A chaque fois qu'on modifie sur le formulaire soit la date d'arrivée soit la date de départ, ou qu'on passe à un autre enregistrement (une autre réservation), MS-Access devrait en principe réexécuter la requête *qselChambresLibres*, pour afficher une nouvelle liste de chambres, qui correspond aux critères modifiés. Comme cette mise-à-jour automatique ne se fait pas de façon automatique, il faudra indiquer à Access le moment de la faire.

   Pour ce faire, nous allons encore une fois nous occuper des événements[1].

   Sélectionnez la propriété *After Update* du champ *txtArrivée*. Cette propriété spécifie en fait un événement, qui est déclenché à chaque fois qu'un changement dans le champ est sauvegardé dans le table *tblRéservations* (p.ex. lorsqu'on quitte le champ après avoir effectué des modifications).

[1] Souvenez-vous que chaque contrôle connaît un certain nombre d'événements prédéfinis, et qu'on peut créer des procédures en Access Basic, qui sont attachées à un événement donné d'un contrôle. Ces procédures sont automatiquement exécutées à chaque fois que l'événement en question est déclenché.

Voici le code déjà attaché à cet événement:

```
Sub txtArrivée_AfterUpdate ()

End Sub
```

**Sub** resp. **End Sub** constituent le début et la fin d'une procédure (voir programmation MS-Excel). Entre ces deux lignes nous pouvons maintenant spécifier notre code Access Basic. Hors, il nous suffit que la requête *qselChambresLibres,* attachée à la zone de liste *lstChambres*, soit réexécutée lorsque l'événement *AfterUpdate* du champ *txtArrivée* est déclenché.

A ces fins, nous disposons de la commande <Nom d'un champ**>.Requery**, qui est capable de réexécuter une requête sur laquelle se base le champ en question

Votre procédure sera donc:

```
Sub txtArrivée_AfterUpdate ()
   lstChambres.Requery
End Sub
```

Procédez de la même façon pour le champ *txtDépart*.

Lorsque l'utilisateur change vers un autre enregistrement, l'événement *On Current* du formulaire même est déclenché. Attachez donc également une procédure contenant la commande **lstChambres.Requery** à cet événement.

5. Testez le formulaire. Vous allez remarquer que pour les réservations existantes, la chambre correspondant à la réservation actuelle n'est jamais affichée. Expliquez ce fait !

6. Proposez une extension à la requête *qselChambresLibres*, qui nous aide à remédier à cet inconvénient.

7. Testez et sauvegardez votre travail

# 11.8  TP No 8 : Les rapports

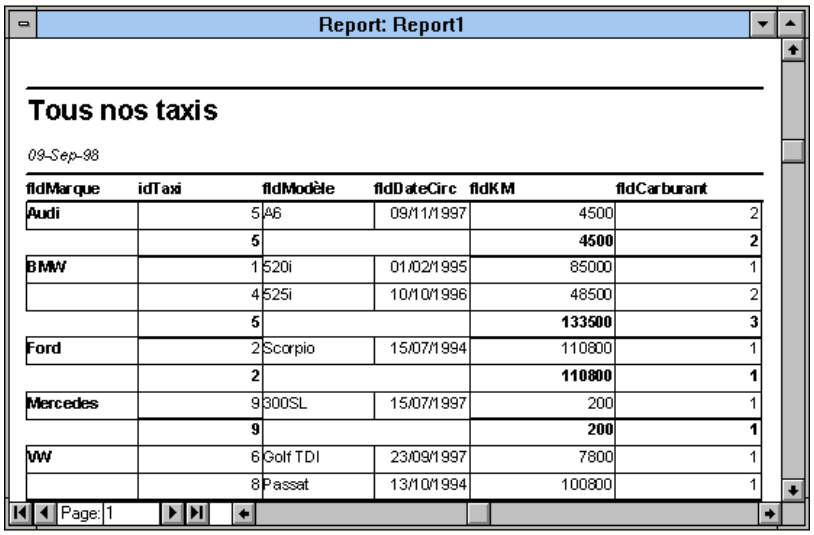
DUREE APPROX : 2h.

**PRESENTATION**

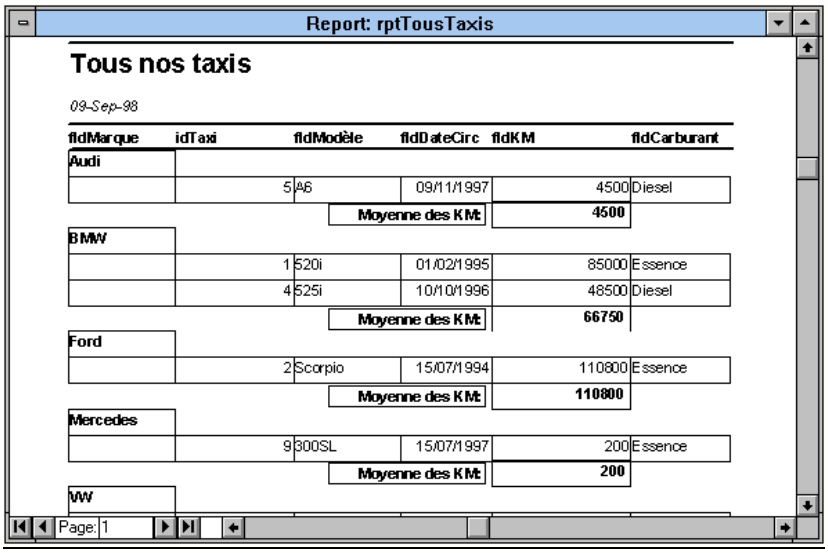
Dans ce TP, nous allons nous baser sur la table *tbltaxis* (*Taxis.mdb*), créée lors du **TP2**.

| Nom du champ | Type de données | Description |
|---|---|---|
| idTaxi | Compteur | Numéro identificateur d'un taxi |
| fldMarque | Texte [15] | Marque du taxi |
| fldModèle | Texte [15] | Modèle du taxi |
| fldDateCirc | Date | Date de la première mise en circulation |
| fldKm | Numérique | Kilométrage actuel |
| fldCarburant | Texte [10] | Essence/Diesel |

D'abord nous utilisons l'"Assistant de Rapport Regroupements/Totaux" (angl. Groups/Totals Report Wizard) pour créer un rapport; qui affiche tous les taxis, groupés par marque. Pour chaque groupe nous allons calculer le total des KM parcourus. Les taxis à l'intérieur d'un groupe (d'une marque) sont triés par modèle.




Ensuite, nous modifions le rapport de façon à ce qu'il convienne exactement à nos besoins, en adaptant par exemple l'emplacement de certains contrôles. En plus, nous remplaçons le calcul du total KM par un calcul de moyenne KM.




Modes

Tout comme pour les autres composants d'une BD, l'utilisateur a le choix parmi plusieurs modes d'affichage pour n'importe quel rapport. Nous avons en principe pour chaque composant un mode structure et un mode utilisation. En ce qui concerne les rapports, Access propose deux modes.

1. Le mode "Création"

   → Ce mode est pareil au mode "Création" des formulaires. On l'utilise pour définir des nouveaux rapports ou modifier des rapports existants. Par analogie aux formulaires, ce mode nous permet de placer les contrôles sur le rapport et de définir les propriétés des contrôles et du rapport.

2. Le mode "Aperçu avant Impression" (angl. Print Preview Mode)[1]

   → Comme les rapports sont généralement imprimés, et ne sont pas utilisés pour des dialogues interactifs avec l'utilisateur, le mode utilisation tel que nous le connaissons pour les autres composants, n'existe pas. Le mode "Aperçu avant Impression" affiche le rapport exactement de la même façon comme il sera imprimé. Ce mode contient en plus toutes les options d'impression (p.ex. Configuration de l'imprimante, Présentation des pages … ).

[1] Access connaît encore le mode "Sample Preview", qui est analogue au mode "Aperçu avant Impression", mais qui n'affiche pas tous les enregistrements disponibles.

## MANIPULATIONS IMPORTANTES

Création d'un rapport

| ACTION | COMMENT FAIRE ? |
|---|---|
| Créer un nouveau rapport en utilisant un assistant | |
| Créer un nouveau rapport sans utiliser un assistant | |
| Sélectionner le mode Création | |
| Sélectionner le mode Aperçu avant Impression | |
| | |

Impression d'un rapport

Afin de pouvoir imprimer un rapport, vous devez sélectionner le mode "Aperçu avant Impression".

| ACTION | COMMENT FAIRE ? |
|---|---|
| Définir les paramètres d'impression | |
| Imprimer le rapport | |
| | |

Manipulation d'un rapport en mode "Création"

| ACTION | COMMENT FAIRE? |
|---|---|
| Afficher la fenêtre Tri et Regroupements (angl. Sorting and Grouping) | |
| Trier sur un champ | |
| Ajouter un groupe | |
| Ajouter une en-tête de groupe | |
| Ajouter un pied de groupe | |
| Activer le générateur d'expression | |
| | |

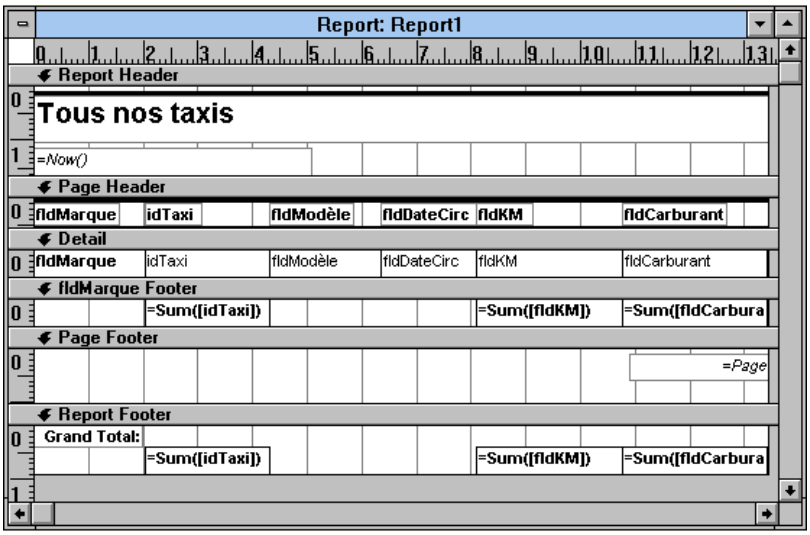
## ACTIVITES

1. Ouvrez la BD *taxis.mdb* .

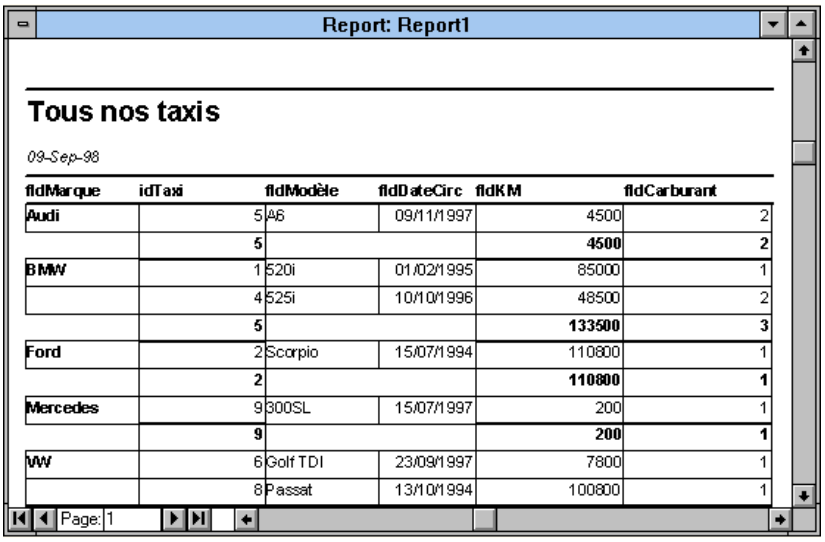
2. Créez un rapport du type "Regroupements/Totaux" (angl. Groups/Totals), basé sur la table *tblTaxis* en utilisant l'assistant correspondant. Voici les indications nécessaires pour répondre aux questions de l'assistant:
   - Le rapport devra contenir tous les champs dans leur ordre original.
   - Groupez les taxis par leur marque (groupement normal).
   - A l'intérieur de chaque groupe, les taxis sont triés par modèle.
   - Utilisez le style prédéfini "Ledger", ainsi qu'une orientation "Portrait".
   - Le titre sera "Tous nos taxis".
   - Sélectionnez l'option qui affiche tous les champs sur une page.
   - Ne sélectionnez pas l'option qui calcule des pourcentages.

3. Sauvegardez le nouveau rapport sous le nom *rptTousTaxis*.

4. Affichez le rapport en mode "Création".




5. Affichez le rapport en mode "Aperçu avant Impression".

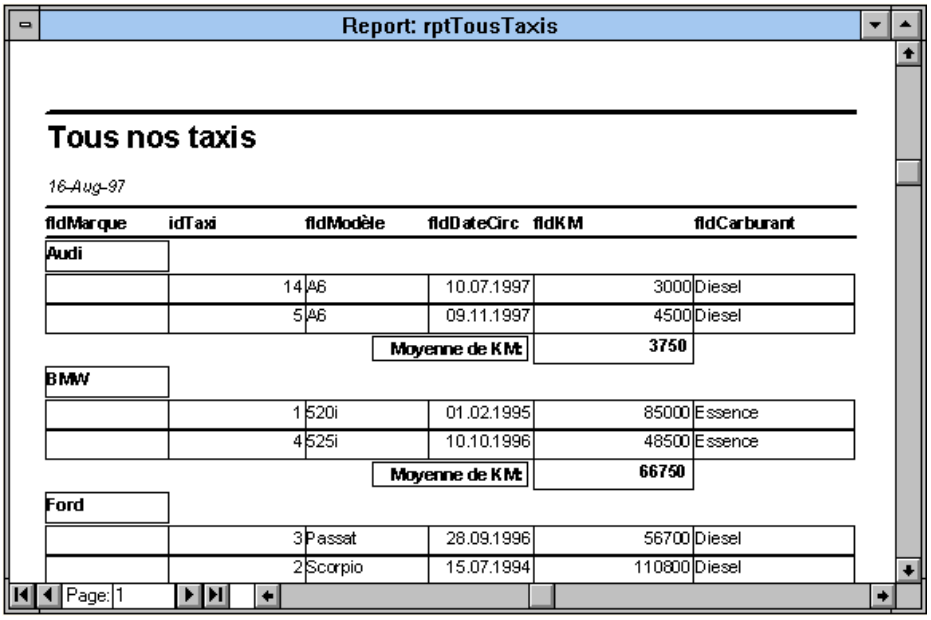



| fldMarque | idTaxi | fldModèle | fldDateCirc | fldKM | fldCarburant |
|---|---|---|---|---|---|
| Audi | 5 | A6 | 09/11/1997 | 4500 | 2 |
| | **5** | | | **4500** | **2** |
| BMW | 1 | 520i | 01/02/1995 | 85000 | 1 |
| | 4 | 525i | 10/10/1996 | 48500 | 2 |
| | **5** | | | **133500** | **3** |
| Ford | 2 | Scorpio | 15/07/1994 | 110800 | 1 |
| | **2** | | | **110800** | **1** |
| Mercedes | 9 | 300SL | 15/07/1997 | 200 | 1 |
| | **9** | | | **200** | **1** |
| VW | 6 | Golf TDI | 23/09/1997 | 7800 | 1 |
| | 8 | Passat | 13/10/1994 | 100800 | 1 |

Est-ce que le rapport créé par l'assistant représente tout à fait ce que vous avez attendu ?

Quels sont les inconvénients ?

6. Nous allons modifier le rapport *rptTousTaxis* de façon à ce qu'il convienne exactement à nos besoins. En plus, nous allons remplacer pour le champ *fldKM*, les calculs des sommes par des calculs de moyennes




7. Affichez le rapport *rptTousTaxis* en mode Création.

Quelles sections contient votre rapport ?

Quelle est la différence entre les contrôles correspondants de la section détail et ceux de l'en-tête de page ?

8. Dans le pied de groupe, supprimez le contrôle qui calcule la somme des numéros de taxi (*IdTaxi*).

   Dans le pied du rapport, supprimez le contrôle qui calcule la somme totale des numéros de taxi (*IdTaxi*).

9. Affichez la fenêtre Tri et Regroupement.

   - A quoi servent les indications de cette fenêtre ?

     _____

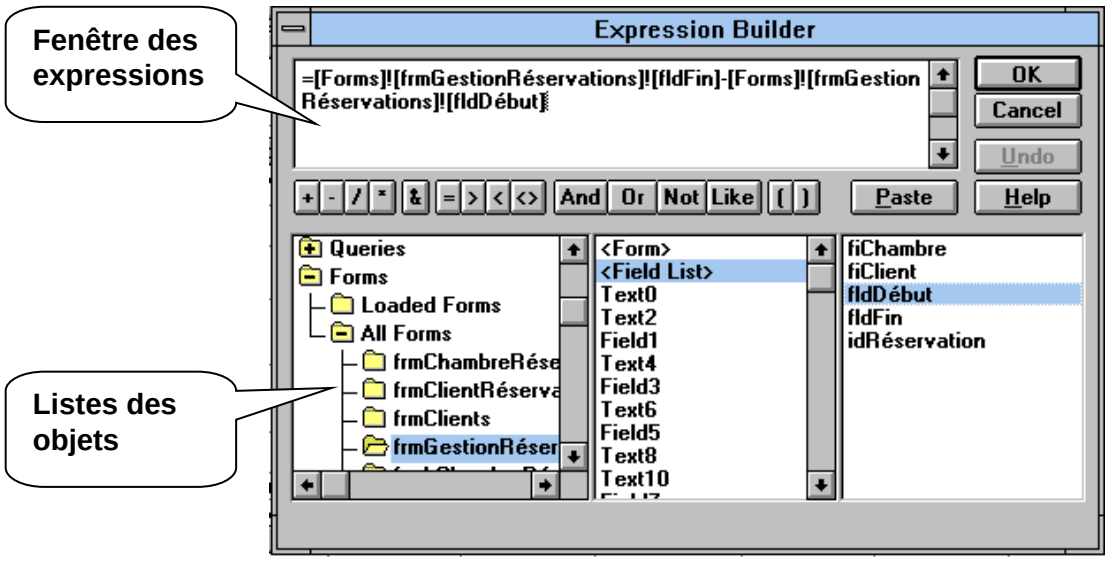
     _____

   - Ajoutez une en-tête de groupe à la groupe *fldMarque*, et déplacez la zone de texte *fldMarque* de la section Détail dans cette en-tête. Affichez le rapport en mode "Aperçu" afin de vérifier l'effet de vos manipulations. Retournez en mode "Création".

10. Dans le pied de groupe, sélectionnez le contrôle qui calcule le total des KM, et affichez les propriétés de ce contrôle[1]. Sélectionnez la propriété *Control Source*. Quelle est l'expression indiquée ?

    _____

    Sélectionnez l'icone [...] afin d'ouvrir le générateur d'expressions.




Les expressions servent à créer des informations qui ne résultent pas uniquement ou même pas du tout des champs des tables de la BD. Utilisées dans les formulaires et rapports, les expressions permettent de calculer la valeur d'un contrôle. Chaque expression peut être une combinaison d'opérateurs, de constantes, de noms de champs, de noms de contrôles et de fonctions.

[1] Pour afficher des propriétés d'un objet, procédez de la même façon que pour les formulaires.

Le générateur d'expressions (angl. Expression Builder) est un outil, qui nous facilite l'écriture d'expressions. Le générateur contient en haut une fenêtre dans laquelle on peut construire une expression, soit en entrant le code de façon manuelle, soit en sélectionnant des objets dans une des trois listes en bas. Il s'agit ici d'une hiérarchie de trois listes dépendant l'une de l'autre avec tous les objets à notre disposition pour créer des expressions. La liste de gauche affiche toutes les catégories d'objets disponibles (Tables, Requêtes, Formulaires, Rapports, Fonctions prédéfinies etc.). Lorsqu'on sélectionne une catégorie, les objets appartenant à cette catégorie sont affichés dans la liste du milieu. Si tel est le besoin MS-Access nous affiche encore des informations dans la liste à droite.

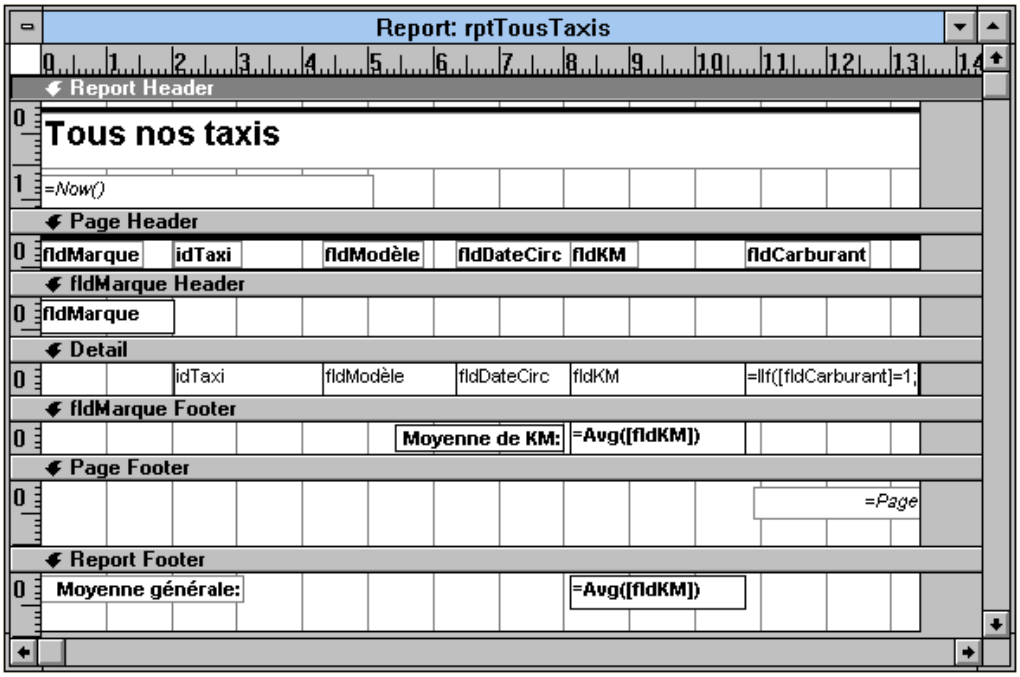
Supprimez l'expression actuelle et remplacez la par une expression qui calcule la moyenne des KM pour les taxis faisant parti d'un groupe. Renommez le contrôle en *Moy_fldKM*. Sélectionnez la propriété *Format* et choisissez le format *Fixed*. Sélectionnez la propriété *Decimal Places*, qui indique le nombre de décimales à afficher, et mettez cette valeur à 0.

11. Dans le pied du groupe créez une étiquette qui affiche le texte "Moyenne des KM:" et placez cette étiquette à gauche du contrôle qui contient l'expression que vous venez de générer. Affichez le rapport en mode "Aperçu" afin de vérifier l'effet de vos manipulations. Retournez en mode "Création".

12. Dans le pied du rapport, modifiez le contrôle, qui détermine la somme totale des KM, de façon à ce qu'il calcule la moyenne des Km pour tous les taxis, et renommez ce contrôle en *GrandMoy_fldKM*. Sélectionnez la propriété *Format* et choisissez le format *Fixed*. Sélectionnez la propriété *Decimal Places*, et mettez cette valeur à 0. Changez le texte de l'étiquette de "Grand Total:" vers "Moyenne générale:".

13. Le carburant est actuellement affiché par son code numérique (1="Essence" et 2="Diesel"). Sélectionnez le contrôle qui affiche la valeur du champ carburant dans la section Détail et affichez les propriétés de ce contrôle. Utilisez le générateur d'expressions pour modifier la propriété *Control Source*, de façon à ce qu'elle contienne une expression conditionnelle (fonction IIF), qui affiche le texte "Essence" lorsque le code numérique vaut 1 et "Diesel" lorsque le code numérique vaut 2. Attention: Vous devez changer le nom du contrôle (p.ex. en *fldCarb*) pour qu'il diffère avec le nom du champ correspondant de la table (*fldCarburant*) pour éviter que MS-Access confonde les deux noms. En effet, dans une expression, le nom du contrôle à priorité sur le nom du champ en cas d'égalité des deux noms.

14. Voici le rapport final en mode Création:




15. Imprimez votre rapport.

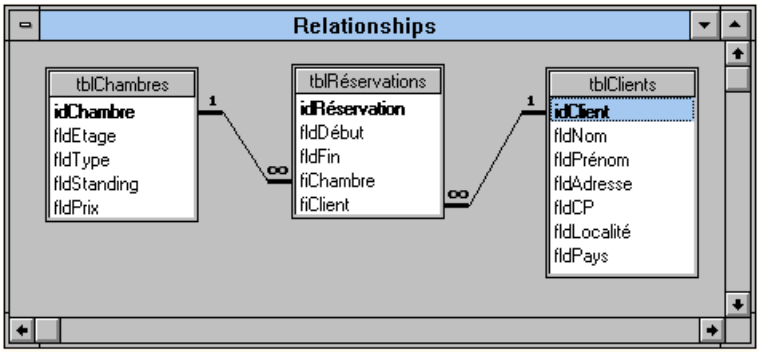
# 11.9 TP No 9 : Les rapports basés sur plusieurs tables

DUREE APPROX : 2h.

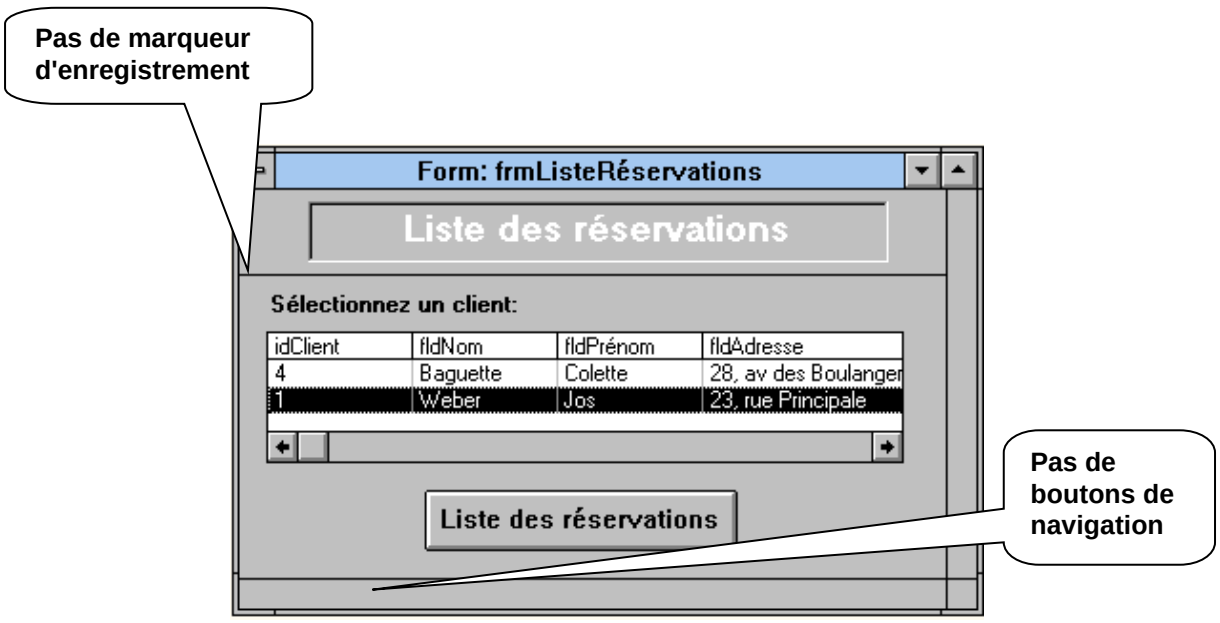
**PRESENTATION**

Le but de ce TP est d'étendre les connaissances en ce qui concerne la création d'un rapport, mais également la combinaison de requêtes, formulaires et rapports pour créer des applications personnalisées.

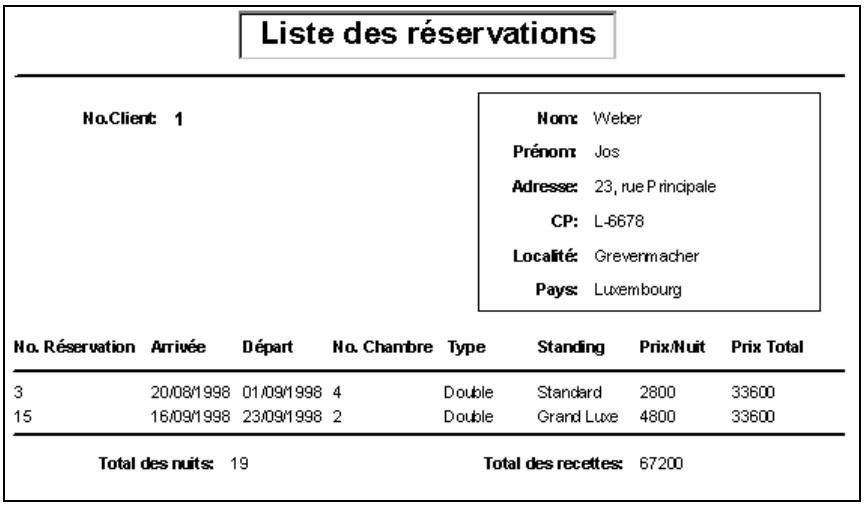
Nous allons nous baser sur les tables suivantes (*Hotel.mdb*), créées lors du **TP4**.




A la fin de ce TP, nous disposons du formulaire suivant.




La sélection du bouton de commande "Liste des réservations" affiche automatiquement un rapport avec les coordonnées du client sélectionné dans la zone de liste et avec les informations concernant toutes les réservations effectuées par ce client.



Liste des réservations

No.Client: 1

Nom: Weber
Prénom: Jos
Adresse: 23, rue Principale
CP: L-6678
Localité: Grevenmacher
Pays: Luxembourg

| No. Réservation | Arrivée | Départ | No. Chambre | Type | Standing | Prix/Nuit | Prix Total |
|---|---|---|---|---|---|---|---|
| 3 | 20/08/1998 | 01/09/1998 | 4 | Double | Standard | 2800 | 33600 |
| 15 | 16/09/1998 | 23/09/1998 | 2 | Double | Grand Luxe | 4800 | 33600 |

Total des nuits: 19 Total des recettes: 67200

Par analogie aux formulaires, il existe des situations où on doit baser un rapport sur une requête, afin de disposer de tous les champs nécessaires pour l'affichage. Le présent rapport en est un exemple parfait, puisque les informations pour chaque réservation proviennent des tables *tblRéservations* et *tblChambres*. Un rapport, tout comme un formulaire ne pouvant être basé que sur un seul objet, il est dans ce cas indispensable de créer une requête, qui contient les champs nécessaires à l'affichage.

**Etapes:**

- Créer la requête sur laquelle sera basé le rapport.
- Créer le rapport.
- Créer le formulaire.
- Adapter le formulaire, en ajoutant un bouton de commande, qui nous permet d'ouvrir automatiquement le rapport avec les données concernant le client actuel.

## ACTIVITES

1. Ouvrez la base *Hotel.mdb* .

2. Nous commençons par la création de la requête *qselClientRéservations*, qui sera à la base du rapport. Cette requête, qui aura comme paramètre le numéro d'un client, affiche pour chaque réservation de ce client:

   - Le numéro de réservation;
   - La date d'arrivée;
   - La date de départ;
   - Le numéro de la chambre concernée;
   - Le type de cette chambre;
   - Le standing de cette chambre;
   - Le prix par nuit de cette chambre;
   - Le prix total de la réservation (Champ à valeur calculée);
   - Le numéro du client;

- Le nom du client;
- Le prénom du client;
- L'adresse du client;
- Le code postal du client;
- La localité du client;
- Le pays ou habite le client.

Créez et testez cette requête.

3. Créez ensuite le rapport *rptListeRéservations* sans utiliser un assistant. Le rapport se base sur la requête *qselClientRéservations*. Quelles sections envisagez-vous dans votre rapport?

 Les réservations d'un client sont triées par ordre ascendant sur la date d'arrivée. Vous devez effectuer ce tri dans le rapport même, puisqu'un rapport en MS-Access ne reprend pas les informations de tri d'une requête.

4. A chaque fois que vous ouvrez le rapport en mode "Aperçu avant Impression", vous devez d'abord indiquer le numéro d'un client, avant de voir les réservations de ce client. Quel est le désavantage de l'entrée manuelle du numéro de client ?

5. La dernière étape de ce TP consiste dans la création du formulaire *frmListeRéservations*, à l'aide duquel l'utilisateur peut sélectionner un client existant, et ouvrir par un clic de bouton le rapport *rptListeRéservations* avec les données pour le client sélectionné. Ce formulaire, qui ne se base sur aucune table, sera créé sans utiliser un assistant.

6. Comme le formulaire lui-même n'affiche pas d'enregistrements (les enregistrements seront affichés dans une zone de liste), vous pouvez enlever les boutons de navigation et le marqueur d'enregistrement. Pour cela, affectez les valeurs NO aux propriétés *Navigation Buttons* et *Record Selectors* du formulaire. Testez l'effet en mode formulaire.

7. Créez le titre dans l'en-tête du formulaire. Désactivez l'assistant des contrôles et insérez une zone de liste dans la section détail du formulaire. Affectez le nom *lstClients* à la zone de liste (au contrôle). Trouvez une méthode pour afficher dans la zone de liste tous les clients triés par ordre alphabétique sur leur nom.

8. En vous servant de l'assistant des contrôles, insérez le bouton de commande sur le formulaire. Donnez à l'assistant les informations suivantes:

   - L'assistant devra ouvrir le rapport *rptListeRéservations* en mode "Aperçu avant Impression";
   - Le bouton de commande devra porter le libellé "Liste des réservations";
   - Le nom du bouton (du contrôle) sera *cmdListe*.

9. Testez le fonctionnement du bouton. Quel en est toujours le désavantage ?

10. Modifiez la requête *qselClientRéservations*; de façon à ce que le paramètre de la requête provienne directement de la valeur actuelle sélectionnée dans la zone de liste *lstClients*.

# 12. Annexes

# 12.1 Bibliographie


| | |
|---|---|
| **[1]** | D.Nanci / B.Espinasse<br>"Ingénierie des systèmes d'information" 3ème Edition<br>Edition: Sybex<br>ISBN: 2-7361-2209-7 |
| **[2]** | P.A.Goupille / J.M.Rousse<br>"Analyse informatique"<br>Edition: Masson<br>ISBN 2-225-84167-5 |
| **[3]** | Marée / Ledant<br>"SQL 2"<br>Edition: A.Colin<br>ISBN: 2-200-2144-1 |
| **[4]** | S.Roman<br>"Bases de données MS-Access / Conception et programmation"<br>Edition: O'Reilly<br>ISBN: 2-84177-054-0 |
| **[5]** | H.F.Korth / A.Siberschatz<br>"Systèmes de gestion des bases de données"<br>Edition: McGraw-Hill<br>ISBN: 2-7042-1170-1 |
| **[6]** | P.Bilke<br>"Start mit Datenbanken und SQL"<br>Edition: KnowWare<br>ISBN: 3-931666-19-4 |
| **[7]** | N.Boudjlida<br>"Bases de données et systèmes d'information"<br>Support de cours<br>Université de Nancy 1 / Faculté des Sciences / Département informatique |
| **[8]** | J.L.Viescas<br>"Running Access 2"<br>Edition: Microsoft Press<br>ISBN: 1-55615-592-1 |

| | |
|---|---|
| **[9]** | Bär / Bauder<br>"Microsoft Access 2"<br>Edition: Micro Application<br>ISBN: 2-7429-0239-2 |
| **[10]** | K.Getz / P.Litwin / G.Reddick<br>"Access 2 Developer's Handbook"<br>Edition: Sybex<br>ISBN: 0-7821-1327-3 |
| **[11]** | CNPI<br>Document de référence "SQL-Matière à traiter"<br>Document de référence "Conception – Vocabulaire" |
| **[12]** | J.M. Jans<br>"La modélisation des données selon la méthode Merise et avec l'outil Win'Design"<br>Support de cours<br>Formation SCRIPT |
| **[13]** | Ministère de la Justice<br>"Projet de Loi relative au respect du droit à la vie privée à l'égard du traitement des données à caractère personnel" & "Exposé des motifs" |
| **[14]** | Ministère de la Justice<br>"Loi du 31 mars 1979 réglementant l'utilisation des données nominatives dans les traitements informatiques"<br>Textes extraits du Mémorial |
| **[15]** | PC Magazin Spezial 5-98<br>"Kryptographie und Netzwerksicherheit" |

## 12.2 Sites sur Internet

w3.restena.lu/proud-online/h_proud.htm

www.pgpi.com

w3.one.net/~jhoffman/sqltut.htm

han2.cbe.wsu.edu/courseware/mis372/slides/Chap08/index.htm

w3.restena.lu/proud-online/ti/infoappl/access/db_prd_0.htm

www.win-design.com/

odyssey.apana.org.au/~abrowne/

www.oracle.com

www.microsoft.com

# 12.3 Index[1]



[1] Les entrées d'index marquées comme propriétés font référence à des propriétés de certains objets de MS-Access, utilisées lors des travaux sur logiciel.

## N



## O



## P



## Q



## R



## S



## T



## U



## V



## Z