# Introduction aux Base de Données et ACCESS

## *PREMISE*

Ce manuel (ou syllabus) ne substitue pas la documentation officielle d'ACCESS™, il est simplement une trace pour apprendre à utiliser ce produite. Pour suivre efficacement ce cours élèves et utilisateurs devraient déjà avoir une connaissance de base de l'ordinateur et d'OFFICE™.

Ce manuel se base sur la version ACCESS 2000, entre différentes versions d'ACCESS il y a des petites différences sur la nomenclature et sur les noms de certains attributs.

## *CONVENCTIONS*

La police Verdana sera utilisée pour indiquer les affichages du menu, ex : Tableau, Propriétés du Tableau.

La syntaxe des commandes, les instructions en langage de programmation et les exemples sont avec police `COURIER NEW`. Dans les explications syntactiques les parties optionnelles sont renfermées entre `[parenthèses carrées]` et les parties variables sont en *`italique`*.

## *Modification de la deuxième édition*

Dans cette édition, a part de la correction de fautes grammaticales et syntactiques et l'efforce de rendre plus lisible le manuel, j'ai ajouté ou amélioré les arguments suivants :

- mettre ensemble (JOIN) les données de plusieurs tableaux,
- créer un état par programmation,
- protection des données et de l'application.

# 1. BASE DE DONNEES

## 1.1 HISTOIRE

Le traitement des informations s'est évolué dans les temps, suivant le développement de la technologie, qu'a mis à disposition des capacités de mémorisation et puissance de calcul toujours croissant. La table suivante est une synthèse de cette évolution.

| AN | ORGANIZATION | FUNCTIONNALITE' | PRODUITS [1] | SUPPORTS PHISIQUES |
|---|---|---|---|---|
| **1950** | **Fichiers** | Access séquentiel | | Cartes perforées, rubans magnétiques |
| **1960** | **Fichiers avec Indexes** | Indexe | ISAM, VSAM | Disques |
| **1970** | **Banque de données (Base de données)** | Indexe, Description des données, langage de programmation | DBASE, CLIPPER, FOXPRO | Disques |
| **1980** | **Banque de données** | Archive avec données, Indexes, description des données, langage de programmation | ACCESS | Disques |
| **1990** | **Serveur pour Banque de données (*DBMS: Data Base Base Management System*)** | Fichiers avec données, Indexes, description des données, langage de programmation, exécution des logiciels par la Base de Données | ORACLE, DB2, SQL SERVER, INGRES, ... | Données distribuées sur disques en réseaux |
| **future** | **Système d'exploitation + DBMS** | | | |

**Figure 1-1** ***Evolution du traitement des informations***

En même temps du développement de la technologie, se sont posé les bases théorique du traitement des informations, qui on porté au model relationnel des données.

Les données, dans leur forme élémentaire, sont un couple nom attribut[2] par exemple:

ville, "BAMAKO",
bitmap, "*le condor du logo*"

ou même nom attributs :

Montagne, "MONT BLANC",Hauteur,4808.[3]

Normalement les données sont groupées pour former des unités logiques, par exemple l'ensemble des données de l'état civil, des attributs d'une image, ou les caractéristiques d'une montagne.
Plusieurs informations élémentaires agrégées sont dites *entité*, parfois record, fiche ou, en terminologie relationnelle, *lignes* (*Rows*). Une donnée générique de la ligne est dite aussi *champ* (*field*). Les entités à son tour sont agrégées en ensembles dits files ou fichier ou archive ou *Tableaux*. Les Base de Données, en anglais Data Base (DB), sont l'ensemble de plusieurs tableaux et d'autres informations corrélées.

L'ordinateur permet de traiter les informations de façon beaucoup plus flexible et rapide que l'élaboration manuelle ; l'ensemble des tableaux, des informations corrélés et d'outils informatiques de gestion comme la gestion automatique des indexes, la sûreté, la programmabilité, etc. il est dit DBMS (*Data Base Management System*).

Les agrégés d'informations peuvent avoir aussi structures complexes, par exemple la fiche d'un employé peut contenir des sous fiches relatives aux augmentations qu'il a eu; si on essaye de faire un schéma de ces structures on obtienne des graphes à arbre:

[1] Les produits cités sont seulement quelquesunes, parmi les plus significatifs, qui ont marqué l'histoire du traitement des données.
[2] Aussi une couple substantif -adjectif ou substantif -nom propre.
[3] Dans la mémoire de l'ordinateur ou sur le support de mémorisation existe seulement l'attribut ; le nom correspond à l'adresse ou à la position du champ dans la structure que le contienne.

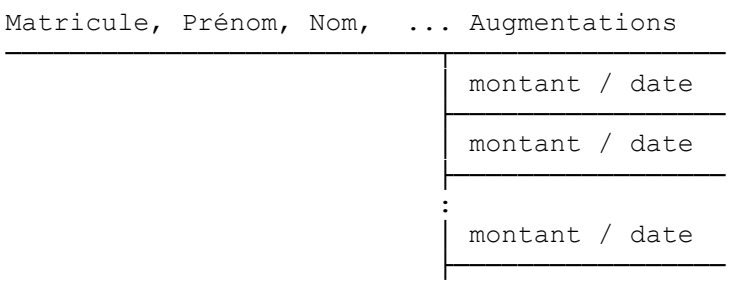


**Figure 1-2** ***Un ensemble hiérarchique***

Les Base de données qui acceptent structures complexes sont dits hiérarchiques ou réticulaires. Puisque l'usage des DB hiérarchiques est complexe, ils ont été remplacés par les Bases de Données Relationnels qui contiennent un ou plusieurs tableaux dans les quels les colonnes sont les champs, et les lignes sont les fiches. La simplicité de la structure relationnelle, d'autre partie, veut une certaine redondance représentée par un champ en commune entre certains tableaux. La représentation relationnelle des données de l'exemple de la Figure 1-2 origines le tableau de l'état civil et le tableau augmentation ; étant la colonne commune la matricule.

| Prénom | Nom | Matricule | ..... |
|---|---|---|---|
| ROSSI | JEANNE | 805567 | ... |
| BIANCHI | ETIENNE | 231784 | ... |
| VERDI | PAUL | 901526 | ... |

**Figure1-3** ***Table état civil***

| Matricule | Montant | Date | ..... |
|---|---|---|---|
| 805567 | 30 | 31/08/1998 | ... |
| 231784 | 20 | 30/04/1996 | ... |
| 901526 | 40 | 30/04/1999 | ... |
| 231784 | 30 | 30/04/2000 | ... |

**Figure 1-4** ***Table Augmentations***

La simplicité conceptuelle des DB Relationnels est évidente dans les opérations qu'on peut faire sur eux à l'aide d'un langage dit SQL (*Structured Query Lanquage*).

Avant de créer un logiciel ou une application[4], on doit faire une activité beaucoup importante, c'est à dire penser à ce qu'on veut obtenir et aux informations dont on à besoin. Cette activité est dite *analyse*; en particulier il faut déterminer les données qui sont nécessaire, le type d'eux (alphanumériques, nombres, date, etc.), leurs dimensions, les valeurs acceptables (*domaine*) et les relations entre eux.

| Nom de l'attribut | Type | Dimension | Domaine |
|---|---|---|---|
| NOM | Caractères | 30 | |
| DATE DE NAISSANCE | Date | | |
| LIEU DE NAISSANCE | Caractères | 25 | |
| SALAIRE | Nombre | | > 20000 et < 100000 |
| ... | | | |
| SEXE | Caractères | 1 | M ou F |

**Figure 1-5** ***Données, type et domaine***

[4] Une application est un ensemble de logiciels pour la gestion d'activités complexes: par exemple la gestion du personnel, la facturation, le contrôle d'un magasin, etc..

## 1.2 LES TABLEAUX RELATIONNELS

### 1.2.1 DÉFINITIONS

Le modèle des bases de données relationnelles a été introduit par Edgar F. Codd, en IBM[5], qu'il a formulé le modèle mathématique et il a spécifié un langage d'interrogation. A' partir de son travail il est né le DB2, une des premières base de données relationnels commerciale. Ici suivent quelques définitions.

- Les *entités* sont déterminées par un ensemble finit de couples `<attribut, valeur>` dit aussi *tuple*, mais il est normalement employé le mot *ligne* (*row*).
  - L'ensemble des attributs est dit *schéma*.
  - L'ensemble des possibles valeurs d'un attribut est dit *domaine*
  - *Relation* est l'ensemble des *tuple* qu'ils ont le même schème, normalement on utilise le synonyme *table*.

En se rapportant à la Figure 1-6, dans la quelle la première colonne est le *schéma* de la Relation Employés, une possible entité pourrais être:

```
NOM                     JEAN VALJEAN
DATE DE NAISSANCE       31/10/1960
LIEU DE NAISSANCE       JEUVILLE
SALAIRE                 5000
...                     ...
SEXE                    M
```

**Figure 1-6 *Exemple d'entité***

Le schéma est: {`NOM, DATE DE NAISSANCE, LIEU DE NAISSANCE, SALAIRE,..., SEXE`}, le domaine de l'attribut `SEXE` est {`M, F`}.

On dit *clé primaire* (*primary key*) un ensemble minimum d'attributs, s'il existe, qu'ils ont valeurs univoques, dans l'exemple la clé primaire pourrais être {`NOM, DATE DE NAISSANCE`}, parce que dans la Relation on peut avoir deux "`ROUGE PIERRE`", mais probablement avec `DATE DE NAISSANCE` différente, toutefois on préfère avoir comme clé primaire un code univoque numérique.

Autres éventuelles clés sont dites *clés secondaires*.

La diction *relationnelle* dérive de la possibilité de mettre en relation entre eux des tableaux différents, en fonction d'attributs communs : dans la table dessous l'attribut `CodeClient` met en relation les tableaux Clients et Ventes, l'attribut `CodeMatériel` lie les tableaux Ventes et Matériels.

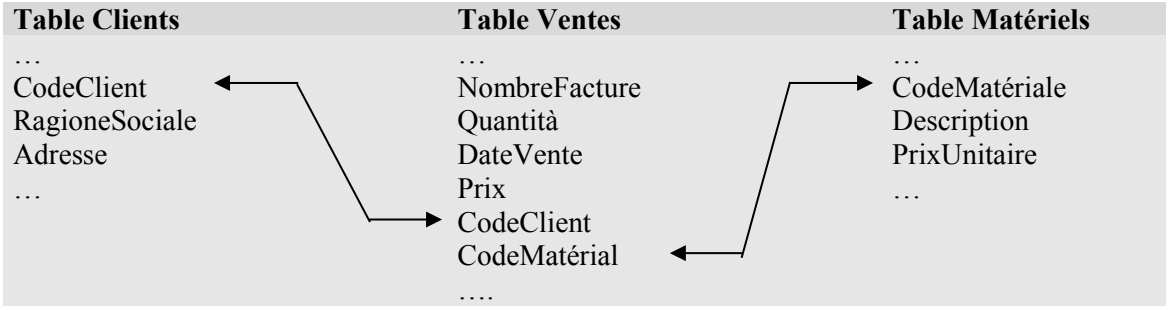


**Figure 1-7 *Relation entre tableaux***

### 1.2.2 OPÉRATIONS

L'opération de *Sélection* permet d'extraire des entités d'une Relation par un prédicat sur les attributs de la relation, par exemple la *Sélection* "`SALAIRE > 5000`" sur la Relation Employés, produise une nouvelle relation qui contienne seulement entités dans les quelles l'attribut `SALAIRE` vaut plus de `5000`.

Entre Relations ayant le même schéma sont possibles les opérations sur ensembles : *Union*, *Intersection*, et *Différence*.

Avec *Projection* d'une relation *R* on entende la relation *R'* obtenue de un sous-ensemble *S* des attributs de *R*.

[5] Edgar F. Codd "A Relational Model of Data for large Shared Data Banks," by Dr. Edgar F. Codd, dans "the Association of Computer Machinery (ACM) journal, Communications of the ACM" June 1970.

Le *Produit* de deux relations $R_1$ et $R_2$ (il est dit aussi *Produit Cartésien*) est une relation dont le schéma est l'union des schémas de $R_1$ et $R_2$ et qui contienne l'union de chaque tuple de $R_1$ avec chaque tuple de $R_2$.

Une *Sélection* sur le *Produit* de deux relations $R_1$ et $R_2$, obtenue par un prédicat de égalité parmi les attributs $A_1$ de $R_1$ et $A_2$ de $R_2$, (attributs ayants le même domaine), est dite *equi-conjonction* ou *equi-join*, et le résultat est une relatione contenant les tuple de $R_1$ unies avec les tuple de $R_2$ pour les quelles vaut $A_1 = A_2$.

## 1.3 LES DBMS

Avec le mot DBMS on indique un logiciel qui gère des Bases de Données (BD) en modalité Client Server c'est-à-dire exécutant des requêtes de données ou de modification de la Base de Données, provenant de plusieurs Clients en même temps.

### 1.3.1 STRUCTURE PHYSIQUE

La structure physique d'une DBMS varie selon la spécifique Base de Données : dans les paragraphes qui suivent il y a quelques exemples.

#### 1.3.1.1 ACCESS

ACCESS techniquement n'est pas un vrai DBMS, même s'il a des aspects d'un DBMS.

ACCESS est une Base de Données qui peut être contenue dans un seul fichier, avec l'application pour la gérer, c'est-à-dire le fichier peut contenir tableaux, formulaires, logiciels etc..

#### 1.3.1.2 MySQL

MySQL est un DBMS gratuite, moyennement puissante et beaucoup utilisé pour des applications Internet.

MySQL peut traiter différentes structures physiques de Base de Données, dont les principaux sont InnoDb et MyISAM. Les Bases de Données correspondent à un répertoire du disque, où, pour les Bases de type MyISAM, il y à les fichiers de description des tables, dont les noms sont les noms des tables avec extensions `.frm`, les fichiers contenants les données, extension `.MYD` et les éventuels fichiers des indexes avec extension `.MYI`. Pour InnoDB il y a seulement les fichiers de description des tables étant données et indexes contenus dans des fichiers (*tablespace*) à part.

Il est important à noter que une Base de Données peut contenir les deux types de structure.

#### 1.3.1.3 Oracle

ORACLE est un des plus puissants DBMS, sa structure est basée sur :

- un ou plusieurs fichiers de données *(Data Files*): ils contiennent les données et toutes les informations relatives à leur élaboration,
- un ou plusieurs fichiers de log *(Redo Logfiles*) : ils contiennent l'histoire des modifications faites à la Base de Données ; ils sont nécessaires pour éventuelles reconstructions de la Base de Données dans le cas de fautes ou perte des données,
- un ou plusieurs fichiers de contrôle (*Control Files*): ils contiennent des informations générales sur la Base de Données : nom, place des archives, etc..

### 1.3.2 STRUCTURE LOGIQUE

Un DBMS peut avoir un ou plusieures Base de données (dans la terminologie Oracle *Tablespace*), et dans eux des objets, *Tableaux*, mais aussi *Requêtes*, *Indexes*, *Logiciels*, *Conteurs*, …
Les *requêtes* ou *Views* ou *Query* sont tableaux "virtuelles", dans le sens que dans le DBMS est mémorisé comme obtenir les données d'une ou plusieurs tableaux ou requêtes mêmes. Comme les tableaux, les requêtes peuvent être interrogées et les données modifiées et ou effacées.

On utilise les requêtes pour :

- cacher la complexité de trouver des données qui sont contenus dans tableaux différents.
- Présenter les informations dans une différente perspective égard à celle du tableau réel.
- Ajouter un ultérieur niveau de sûreté, en limitant l'accès à certaines lignes et colonnes d'une ou de plusieurs tableaux.

Les *Indexes* sont la composante de Banques de Données qui permettent l'accès rapide à des informations.

*Les Logiciels* sont la composante "personnalisé" du DBMS, ils sont fonctions, procédure ou élaborations que le DBMS effectue au vérifier d'évents particuliers (*trigger*).

Il est à remarquer que pratiquement toutes les informations qui servent au DBMS pour sa gestion sont maintenues dans des tableaux qui sont la Base de Données du système.

### 1.3.3 LE MOTEUR DU DBMS

Les DBMS ne sont pas simplement des fichiers d'informations, mais ils sont des gérants d'informations : dans un environnement multi usagers ils permettent l'accès en contemporanéité aux données, en assurant un haut niveau de prestation, l'intégrité des données et des solutions efficaces pour faire front aux fautes dues à dégât physique. Les DBMS exécutent aussi des taches prédéterminées et automatiques, comme le journal des activités, la gestion des indexes et des espaces, l'exécution d'instructions liées à certains évents (*trigger*, fermeture du travail…), etc..

## 1.4 LE "LANGAGE" SQL

SQL est devenu le standard pour accéder aux données contenues dans les Bases de Données relationnelles. Techniquement SQL il n'est pas un langage de programmation complet, ils manquent en effet des instructions pour conditionner l'exécution ; il est surtout un outil pour interagir avec un BD relationnelle à niveau logique, avec une haute abstraction de l'implémentation des données.

Entre les possibilités de SQL on a :

- création, modification et effacement d'objets,
- interrogation de données, le résultat est une relation,
- insertion, modification et effacement de lignes.

Ici il y a seulement un aperçue de SQL selon la syntaxe acceptée par ACCESS ; il faut noter, toutefois, que ACCESS contient des outils que créent les nécessaires command pour la création des tableaux, des requêtes et de la mise à jour des données.

### 1.4.1 CRÉATION D'UNE TABLE

Pendant la création d'une table, outre à déterminer les champs et leur format, il est possible indiquer si celui ci est une clé et de quel type.

ACCESS (et les DBMS) acceptent beaucoup type de donnés[6], ici une petite liste des plus communes :

| Type | Description |
|---|---|
| `INTEGER, LONG` | Pour mémoriser des nombres entier |
| `MONEY` | Pour mémoriser des nombres avec quatre décimaux |
| `AUTOINCREMENT` | Nombre entier qui s'auto incrément pour chaque nouveau ligne |
| `VARCHAR,TEXT` | Pour les champs de texte jusqu'à à 255 caractères, ex. `TEXT(12)` |
| `DATE` | Pour mémoriser des dates et heures |
| `MEMO` | Pour mémoriser texte de longueur supérieure à 255 ou données non textuels comme images, sons, etc. |

**Figure 1-8** ***Types de donnés***

Exemples :

```
CREATE TABLE agents (
  IdAgent    AUTOINCREMENT PRIMARY KEY,
  Agent      VARCHAR(30) NOT NULL,
  IdResp     INTEGER,
  Montant    DOUBLE
  )

  CREATE TABLE ventes (IdV AUTOINCREMENT PRIMARY KEY, IdAgent INTEGER,
                       Montant DOUBLE, Ville TEXT(25) NOT NULL)
  CREATE TABLE villes (IdAgent INTEGER,Ville TEXT(25) NOT NULL)
```

**`PRIMARY KEY`** engendre un index univoque et pas nulle sur la colonne *nom*. L'indication **`NOT NULL`** de la colonne *Agent* indique que la colonne ne pourra pas être vide.

[6] SQLite est un simple logiciel pour gérer des Bases de Données où tous données sont textuels, sauf l'éventuel **`PRIMARY KEY`**.

Pour modifier une table le command est `ALTER TABLE nomdetable ...` :

Exemples :

- Ajouter un champ : `ALTER TABLE clients ADD COLUMN Telephone TEXT(20)`
- Modifier un champ : `ALTER TABLE clients MODIFY COLUMN Nom TEXT(30) NOT NULL`
- Effacer un champ : `ALTER TABLE clients DROP COLUMN Province`

L'instruction pour éliminer une table est : **`DROP`** `TABLE nomdetable`.

### 1.4.2 LES INTERROGATIONS

La figure suivante montre la structure du commando `SELECT` de SQL qui opère sur une ou plusieurs relations :

| **SELECT** | **Schema** | **projection** |
|---|---|---|
| FROM | Relation(s) (*Table*) | Introduise la (les) relation(s) |
| WHERE | Condition(s) | Sélection |
| GROUP BY | Attribut(s) | Groupement d'attributs |
| ORDER BY | Attribut(s) | Ordre de présentation des données |

**Figure 1-9** ***Structure du commande SELECT***

Seulement les clauses `SELECT` et `FROM` sont nécessaires.

Le command SELECT est suivi des données qu'on veut voir ; celui-ci peut être :

- `*` (astérisque) tous les champs,
- une liste de champs et/ou de fonctions sur eux, séparés par `,` (virgule).

Les fonctions acceptées par ACCESS sont les mêmes de Visual Basic.

La commande visualise les données voulus avec un entête formé par le nom du champ, toutefois est possible indiquer un autre nom, surtout pour les champs calculés, avec la syntaxe suivante :

```
SELECT ... nom AS entête, ...
```

| | |
|---|---|
| Tous les champs | `*` |
| Quelques champs | `Nom, Adresse, Ville, ...` |
| Valeurs calculées | `..., Montant * 1,20, ...` |
| Champs avec entête de l'usager | `..., Montant * 1.20 AS [MONTANT + IMPOT], ...` |

**Figure 1-10** ***Exemple de schéma***

Exemples :

- `SELECT * FROM anag`
- `SELECT nom, adresse, Montant * 1.20 AS [Montant lourd], Préfixe + '/' + téléphone AS télé FROM anag`

La clause FROM`FROM` indique le tableau ou les tableaux dont on veut les données. Le tableau peut être aussi engendrée par un command `SELECT`. La syntaxe est :

```
SELECT ... FROM relation1[[AS] alias1],relation2[[AS] alias2],...
```

Où $relation_n$ est le nom d'un tableau, d'une requête ou une `SELECT`.

Quand il y a plusieurs tableaux on peut indiquer des champs de tous les tableaux. Dans la liste des champs de `SELECT` les ambiguïtés (nom de champs égales dans les différents tableaux), sont évité par la syntaxe :

```
SELECT ...,nomdetable.nomdechamp, ... FROM ...,nomdetable,...
```

Où *nomdetable* est le nom du tableau ou le nom assigné par la clause `AS` $alias_n$.

**Exemples**

- `SELECT nom, anag.province, chieflieu FROM anag, tab_pro`
- 
```
SELECT F.nom,Quantitè FROM farm AS F,
    (SELECT nom, sum(qty) AS Quantitè FROM Med GROUP By Nom) AS M
WHERE F.nom = M.nom
```

La clause WHEREAvec la clause `WHERE` on choisit les lignes du tableau que on veut voir, la clause est suivie par une expression qui est évalué pour toutes les lignes du tableau ; si la relation est vraie la ligne sera choisie.

- ```
  SELECT nom, anag.province, chieflieu FROM anag, Tab_pro
  WHERE anag.province = Tab_pro.province
  ```

Dans expression les constants caractères sont entre apostrophes.

Les opérateurs de confrontation sont =, >, <, >=, <=, `LIKE`, `IN`. Les opérateurs logiques `AND`, `OR` et `NOT` sont utilisable dans la condition :

- `condition`$_1$ `AND condition`$_2$ sera vraie si tous les deux conditions sont vrais,
- `condition`$_1$ `OR condition`$_2$ sera vraie si au moins une condition est vraie,
- `NOT condition` sera vraie si la condition est fausse.

`LIKE` est utilisé pour des confrontations avec le contenu partiel des champs de texte, à l'aide de caractères de masque `*` et `?` (*wild card*)[7] :

- `*` quelconques ensembles de caractères
- `?` un caractère : ex. ?oss

ex. `SELECT * FROM ANAG WHERE Nom LIKE '?oss*'` trouve Rossati, Rossi, rossini, Fossati, etc.

Examples:

```
SELECT * FROM ANAG WHERE Province IN ('AL','NO')
SELECT * FROM ANAG WHERE Province NOT IN ('AL','NO')
SELECT * FROM ANAG WHERE Quantité > 4 AND Quantité  < 11
```

### 1.4.2.1 La clause ORDER BY

Pour obtenir les données ordonnées, même avec plusieurs champs ou expression, séparé par virgule. Le mot DESC après le nom, indique un ordre descendant :

```
SELECT * FROM MED order by nom,qty DESC
```

### 1.4.2.2 La clause GROUP BY

La clause `GROUP BY ...` sert pour obtenir des donnés synthétisées. Dans le groupement peuvent seulement paraître les champs sur les quels se font les groupements et/ou des fonctions d'agrégation : `AVG`, `COUNT`, `MAX`, `MIN`, et `SUM`.

- ```
  SELECT provincia, Count(Provincia) AS
  'N.', SUM(quantita) 'TOTAL PROVINCE'
  FROM anag GROUP BY provincia
  ```

```
PR         N. TOTAL PROVINCE
-- --------- --------------
CN          2            2.5
PD          1            2.7
RM          1            2.2
TO          1            1.5
VR          2            1.9
```

La liste aura implicitement l'ordre des champs de `GROUP BY`, a moins de la présence de la clause `ORDER BY`.

- ```
  SELECT bureau, MIN(salaire), MAX(salaire)
  FROM emp GROUP BY bureau ORDER BY
  MIN(salaire)
  ```

Il est possible choisir des données du tableau obtenue, par la clause `HAVING`, qui est analogue à la clause `WHERE` : `WHERE` agisse sur la table original, `HAVING` sur les résultats groupés.

```
SELECT Nom,SUM(Qty) AS Total FROM med
WHERE Lotte > 0
GROUP BY Nom HAVING SUM(Qty) > 10
ORDER BY SUM(Qty) DESC
```

Ici à gauche un exemple complet.

L'exemple ci-dessous montre comme ajouter des totaux à l'aide du command SQL `UNION` (`UNION` veut des ensembles de données avec la même structure) :

[7] Les *wild card* d'ACCESS ne sont pas standard, dans MySQL et Oracle elles sont respectivement `%` et `_`.
```

```
SELECT Lotte,Nom, Total FROM (
    SELECT Lotte AS LotteX, Lotte, nom, SUM(qty) AS Total
      FROM Med
      GROUP BY Lotte, nom
  UNION
    SELECT Lotte & 9 AS LotteX, 'Total', '' AS nom, SUM(qty) AS Total
      FROM Med
      GROUP BY Lotte
  UNION
    SELECT 99999 AS LotteX,'Total General' AS Lotte,'' AS nom, SUM(qty) AS Total
    FROM Med)
    XXX
ORDER BY Lottex,Nom
```

**Figure 1-11** ***SELECT avec totaux***

### 1.4.2.3 Données de plusieurs tableaux

L'opération qui produit une liste avec données appartenant à plusieurs tableaux (ou requêtes) est appelée *Join*, et elle est réalisé par la clause `JOIN`, dans laquelle il y a une condition, habituellement d'égalité, sur des champs des tableaux impliqués. Il y a plusieurs formes de JOIN, selon ce que on veut obtenir, qui sont essentiellement :

- données correspondants,
- tous données qui appartiennent à une table avec ce qui correspondent dans l'autre table,
- les données d'une table qui n'ont pas des correspondants dans l'autre,
- tous données avec ou sans correspondance dans les tableaux.

L'exemple à droite liste l'agent responsable pour chaque ville ou il y a eu des ventes.
On note que le premier tableau est une vue matérialisée, à la quelle on a donné le nom `Statistique`.
On peut lister quelconques colonnes ou expression des deux tableaux ; pour éviter les ambiguïtés des noms de colonnes égales on doit qualifier celui ci par le nom du tableau, avec l'écriture : *`nom_detable.nom_de_colonne`*.

```
SELECT Statistique.Ville, Agent, Montant
  FROM (SELECT Ville, SUM(Montant) "MONTANT"
        FROM Ventes
        GROUP BY Ville) Statistique,Villes
  WHERE Statistique.Ville = Villes.Ville;
VILLE                AGENT                   MONTANT
-------------------- -------------------- ----------
Buja                 Condor                      675
Bukavu               Wamesa                       57
Kiremba              Luciu                       500
Ngozi                Rafiki                      250
```

#### 1.4.2.3.1 DONNÉES CORRESPONDANTS

La syntaxe est : `SELECT ... FROM` *`table1`* `INNER JOIN` *`table2`* `ON condition ...`, example :

```
SELECT agents.*,Ville FROM agents INNER JOIN Villes ON agents.idagent =
villes.idagent ORDER BY montant Desc
```

| IdAgent | Agent | IdResp | Montant | ville |
|---|---|---|---|---|
| 5 | Jaune | 2 | 37000 | Marseille |
| 1 | Rouge | 0 | 34000 | Turin |
| 10 | Brun | 3 | 32500 | Madrid |
| 9 | Beige | 1 | 29800 | Seville |
| 2 | Vert | 1 | 23000 | Milan |
| 4 | Gris | 3 | 21000 | Nice |
| 11 | Carmin | 2 | 18700 | Barcelone |
| 7 | Blanc | 6 | 14000 | Toulon |
| 8 | Azur | 2 | 12500 | Palerme |
| 3 | Bleu | 1 | 12000 | Geneve |

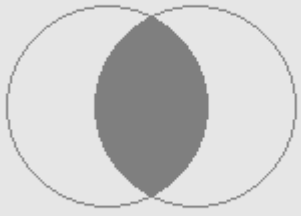

**Figure 1-12** ***Inner JOIN***

Seulement dans ce cas on pourrait utiliser le command qui utilise la clause `WHERE` :

```
SELECT agents.*,ville FROM agents,Villes WHERE agents.idagent =
villes.idagent ORDER BY montant Desc
```

#### 1.4.2.3.2 TOUS DONNÉES QUI APPARTIENNENT À UNE TABLE ET CORRESPONDANTS

La syntaxe est : `SELECT ... FROM` *table*$_1$ `LEFT JOIN` *table*$_2$ `ON condition ...`, dans ce cas tous données de *table*$_1$ seront choisi avec éventuels données correspondent dans *table*$_2$. Si on veut le contraire ou on change *table*$_2$ *avec* *table*$_1$ ou on utilise la clause `RIGHT JOIN`, exemple :

```
SELECT agents.*,Ville FROM agents LEFT JOIN Villes ON agents.idagent =
villes.idagent ORDER BY montant Desc
```

| IdAgent | Agent | IdResp | Montant | Ville |
|---|---|---|---|---|
| 5 | Jaune | 2 | 37000 | Marseille |
| 1 | Rouge | 0 | 34000 | Turin |
| 10 | Brun | 3 | 32500 | Madrid |
| 9 | Beige | 1 | 29800 | Seville |
| 2 | Vert | 1 | 23000 | Milan |
| 4 | Gris | 3 | 21000 | Nice |
| 11 | Carmin | 2 | 18700 | Barcelone |
| 7 | Blanc | 6 | 14000 | Toulon |
| 13 | Rose | 6 | 13000 | |
| 8 | Azur | 2 | 12500 | Palerme |
| 3 | Bleu | 1 | 12000 | Geneve |
| 6 | Noir | 1 | 9000 | |

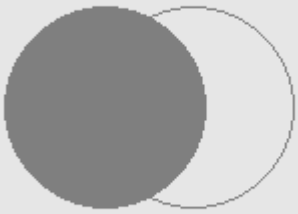

**Figure 1-13 *LEFT JOIN***

#### 1.4.2.3.3 DONNÉES D'UNE TABLE SANS CORRESPONDANTS

On utilise la clause `LEFT JOIN`, où la première table est celle dont on veut le contrôle, vérifiant l'inexistence d'un champ de la deuxième table :

```
SELECT ville FROM Villes LEFT JOIN
Agents ON agents.idagent =
villes.idagent WHERE
ISNULL(agents.idagent)
```

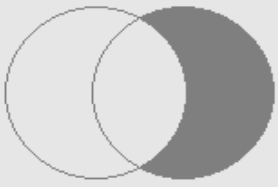

**Figure 1-14 *LEFT JOIN sans correspondants***

#### 1.4.2.3.4 TOUS DONNÉES

On peut obtenir les données de deux tableaux avec `l'UNION` de deux requêtes :

```
SELECT agents.*,Ville FROM agents LEFT JOIN Villes ON agents.idagent =
villes.idagent UNION SELECT agents.*,ville FROM Villes LEFT JOIN Agents ON
agents.idagent = villes.idagent WHERE ISNULL(agents.idagent)
```

#### 1.4.2.3.5 JOIN SUR LA MÊME TABLE

Le JOIN sur la même table est indiquée comme *self join* :

```
SELECT W.Agent AS Responsable, V.Agent AS Agent, V.Montant AS Montant FROM
Agents V LEFT JOIN Agents W ON W.IdAgent = V.IdResp ORDER BY W.Agent
```

| Responsable | Agent | Montant |
|---|---|---|
| | Rouge | 34000 |
| Bleu | Brun | 32500 |
| Bleu | Gris | 21000 |
| Noir | Rose | 13000 |
| Noir | Blanc | 14000 |
| Rouge | Beige | 29800 |
| Rouge | Noir | 9000 |
| Rouge | Bleu | 12000 |
| Rouge | Vert | 23000 |
| Vert | Carmin | 18700 |
| Vert | Azur | 12500 |
| Vert | Jaune | 37000 |

**Figure 1-15** ***JOIN su la même table***

#### 1.4.2.4 Croisement de Données

Dans SQL d'ACCESS, il y a la possibilité de produire des données croisé avec la commande `TRANSFORM` :

`TRANSFORM` *aggregation* `SELECT` $champ_1$ `FROM` *source* `GROUP BY` $champ_1$ `PIVOT` $champ_2$

dont ici suit un exemple :

```
TRANSFORM Sum(Montant) SELECT Agent FROM rqtVentes GROUP BY Agent PIVOT Ville
```

Qui donne quelque chose d'analogue :

| Agent | Geneve | Milan | Naples | Rome | Turin |
|---|---|---|---|---|---|
| Bleu | | 21000 | | 32500 | |
| Noir | | 13000 | 14000 | | |
| Rouge | 29800 | | | 35000 | 43000 |
| Vert | | | 37000 | | 31200 |

### 1.4.3 MODIFICATION DES DONNÉES

#### 1.4.3.1 Insertion de nouveaux données dans une table

```
1) INSERT INTO anag (nom, adresse, cap, province, ville)
       VALUES ('ROSSATI Giovanni', 'Oriani 3', '10149', 'TO','Torino');
2) INSERT INTO anag (nom, ville, province, cap)
       SELECT nom, ville, province, cap FROM anag2
       WHERE province = 'TO';
```

L'exemple **2)** "copie" les colonnes *nom*, *ville*, *province* du tableau *anag2* appartenant au lignes qui satisfint à : `WHERE province = 'TO'` et les insère dans le tableau *anag*.

#### 1.4.3.2 Variation des données d'une table

```
1) UPDATE anag SET quantite = 1.5 WHERE nom like '?oss*'
2) UPDATE anag SET adresse = 'Castelrotto 66', CAP = '23023'
       WHERE nom = 'Franchi Elmo';
```

**Attention** : si on omit la clause `WHERE` toutes les lignes seront changé.

Effacement des données d'une tableau

```
1) DELETE FROM anag WHERE nom = 'Fossati'
2) DELETE FROM anag
```

L'exemple **2)** efface toutes les données contenues dans le tableau *anag*, le tableau existe encore.

Pour éliminer un tableau le command SQL est : **`DROP`** `TABLE` *nomdetable*.

# 2. ACCESS PRATIQUE

Access est un logiciel pour mémoriser et élaborer informations structurées. Access aide l'utilisateur dans toutes les phases que regardent le traitement des données : du projet des tableaux, à la réalisation de formulaires pour la gestion des données, jusqu'à la préparation des états. Le langage Basic permet d'ajouter des fonctionnalités à l'application.

Beaucoup fonctions sont exécuté avec l'aide de « Assistants », et un aide en ligne est disponible dans tous contextes par la touche F1.

Les principaux objets d'ACCESS sont :

- Tableaux : sont la place où les informations sont contenues; dans la définition des tableaux on détermine les informations contenues, avec ses caractéristiques, s'il y a des contrôles sur les valeurs, l'éventuelle valeur de défaut.
- Requêtes ou *query*: il est une façon de voir ensemble les données d'une ou plusieurs tableaux, par exemple la facture avec les données du tableau des ventes, des clients et du magasin. Les données des requêtes peuvent aussi être le résultat d'opérations sur les données d'un tableau, par exemple si le tableau contient les champs `prix` et `escompte`, la requête peut contenir `prix * escompte`.
- Formulaires : elles sont des interfaces graphiques pour organiser l'application, visualiser les données des tableaux ou des requêtes et pour les gérer, c'est-à-dire pour modifier, effacer et insérer des nouveaux données.
- Etats ou *report* : ils sont la description pour obtenir l'états des données d'un tableau ou d'une requêtes, avec éventuels triage et totalisations.
- Modules : ils permettent d'ajouter des fonctions particulières aux fonctions natives d'ACCESS. Les modules sont écrits avec le langage de programmation VBA (), qu'il est aussi utilisé pour gérer les évents des formulaires, et des états.

Au démarre de Access, il propose :

- Nouvelle base de donnée Accces,
- Assistant, pages ... pour choisir entre diverses projets-exemples,
- Ouvrir un fichier existant.

Ils existent en ACCESS deux modalités de travail : *exécution*, qui est le mode de travail de l'utilisateur de l'application, et mode *projet* qui permet de créer une application en gérant les composants de la Base des Données, c'est-à-dire tableaux, requêtes, formulaires pour l'introduction des données, états et programmation. Dans ce chapitre on examinera la modalité *projet* d'ACCESS.

La création d'une application est géré par des formulaires et pour la plupart des composants d'ACCESS est disponible une construction conduite (Assistant).

Il y a une fenêtre pour gérer les objets d'ACCESS, choisissant un objet il est possible choisir entre la création d'un nouveau objet ou sa modification.

## 2.1 TABLEAUX

Il est possible créer un nouveau table, soit en indiquant les champs dans la fonction Visualisation structure , soit en copiant, et éventuellement en modifiant, une structure de table préexistant.

Type des champs En Visualisation structure il faut indiquer obligatoirement les noms des champs et leur type. Cette dernière indication est choisie par un menu à descente qu'il est accessible en cliquant sur le champ Type de donnée. Ici en bas une aperçue des types de champs.

| type | utilisé pour | caractéristiques |
|---|---|---|
| texte | adresses, noms, codes, même si numériques, ... | de 0 aux 255 caractères |
| numéro | montants, quantité, pourcentages, … | entiers ou avec des chiffres décimaux |
| date/heure | dates et heures | |
| compteur | matricules, codes clients, matériels, factures. | le champ est géré automatiquement et il sert à numéroter les fiches |
| mémo | textes, documents, images. | |
| oui/non | oui/non, vrai/faux, absent/présent. | |

**Figure 2-16 *Type des champs***

Entre les choix, il y a aussi Assistant Liste de choix utile pour les champs qui peuvent contenir un numéro limité de valeurs (Sexe, Scolarité, Service, etc.). Avec l'assistant on indique ou une liste de valeurs ou un tableau qui contient les valeurs ; on verra que dans les formulaires, au lieu de saisir les valeurs, ils pourraient être choisi d'une liste déroulant .

Au-delà de nom et type, il est possible d'indiquer autres attributs du champ, ceux-ci servent à faciliter l'introduction et le contrôle des données. Ces attributs dépendent du type de donnée.

Quand un des attributs est sélectionné, il peut apparaître à droite un bouton de liste pour permettre un choix entre différentes alternatives ou un bouton avec … qui fait apparaître une fenêtre de dialogue.

| attribut | utilisé pour |
|---|---|
| Dimension | Indiquer la dimension maxime d'un champ texte ou numérique |
| Forme | presque pour toutes les types, utile pour dates et montants |
| Masque d'input | Pour faciliter l'introduction des données (surtout pour les dates) |
| Étiquette | Aide pour l'input des données |
| Valeur prédéfinie | Pour introduire une valeur prédéfinie |
| Valide si | indique un contrôle formel |
| Message pour faute | Utilisé pour signaler l'introduction de données erronées |
| Obligatoire | Impose l'introduction du champ |
| Permets longueur zéro | |
| Indexé | Cree un indexe pour rendre vites les recherches et les imprimeries |

**Figure 2-17** ***Attributs des champs***

**Remarque** Le champ `Étiquette` est utilisé pour donner l'étiquette aux champs des formulaires et des états, ceci permet de développer vitement et utiliser des dictions homogènes.

### 2.1.1 CONTRÔLES SUR LES CHAMPS

Dans les champs Valide si il est possible d'insérer une expression de contrôle direct ou une expression avec des fonctions de *VBA* ou des fonctions de contrôle écrites avec *VBA* par le programmateur. L'expression de contrôle peut être composée avec l'aide du "Générateur d'expressions", ceci on l'obtient avec clic sur le bouton qui apparaît à droite du champ "Valide si", quand celui-ci est sélectionné.

| | |
|---|---|
| `Date()+10` | Date du jour plus 10 |
| `>= 65` | Valeur majeur ou égale de 65 |
| `LIKE 'ROSS*'` | Accepte noms qui commencent par ROSS |
| `'M' Or 'F'` | Accepte seulement M ou F |

**Figure 2-18** ***Exemple de contrôles sur champs***

Un éventuel contrôle d'unicité on l'obtient en indiquant le champ comme indexé et en choisissant l'option Oui duplicatas pas admis.

### 2.1.2 LES INDEXES

Les index sont nécessaires pour grandes tables, il est possible d'introduire index formé par plusieurs champs et même sur des expressions, par la fonction de menu Affichage et puis Index. La balise avec la clé  indique Primary key c'est à dire un index qu'il doit être présent et univoque ; normalement est de type `compteur automatique`.

## 2.2 FORMULAIRES

Les Formulaires sont les objets qui permettent de gérer les données par introduction guidée. ACCESS permet de créer des formulaires personnalisés ou avec une Création guidée. Les formulaires sont associés, normalement, à un tableau ou à une requête. Les formulaires si obtenus sont douées par défaut, des butons pour créer, modifier, effacer et rechercher des donnés.

### 2.2.1 LES OBJETS DES FORMULAIRES

Les *formulaires* sont des conteneurs d'objet pour insérer de données (zone de texte, zone de liste, zone de liste modifiable ou *combo box*), radio buttons, check box ou pour les gérer (buttons, barre de déroulement, etc..) ou pour obtenir des effets esthétique (images, lignes, fonds, etc..).
A chaque objet sont associé des propriétés qui sont accédé par :

- double clique sur l'objet ou
- clique droit et choix Propriété ou
- clique sur la balise des propriétés :  .

### 2.2.2 CRÉATION D'UN FORMULAIRE

On peut engendrer un formulaire par la fonction Créer un formulaire a l'aide l'Assistant ou le bâtir soi-même déplaçant les objects sur la fenêtre. Les objets sont choisis dans la Boite a outils (Menu Affichage, Boite a outils) ou la balise.

- **Texte** est utilisé pour insérer du texte ou visualiser des informations, les textes peuvent être associés à des champs d'une table ou être engendré par une formule.
- **Liste** et **Liste déroulant** pour choisir des données entre un ensemble fixe de valeurs, la liste déroulant peut aussi permettre l'introduction d'un texte pas présente dans la liste (propriété Limiter à list No). Si dans la boite des outils la balise est choisie, une procédure conduite permettra d'indiquer si les données sont dans une Table ou sont obtenue par une Requête ou sont indiqué directement.
- **Bouton** pour exécuter des commandes, si la balise est active, on est conduit sur des choix entre plusieurs actions et il est automatiquement engendré le code VBA pour la gestion.
- **Case à cocher** et **Bouton d'option**.

L'Assistant engendre un formulaire pour insérer les valeurs des champs comme zone de texte ou de case à cocher pour les données de type Oui/Non et des listes déroulantes dont on à utilisé l'Assistant Liste de choix ; le formulaire est engendré avec des boutons standard pour le déplacement entre les données.

Propriété et événements des formulaires On accède aux propriétés et événements des formulaires en sélectionnant le formulaire avec un click sur le bouton en haut à gauche au croisement des règles. Ici de suite quelques propriétés.

| propriété | valeur | caractéristiques |
|---|---|---|
| Étiquette | texte | titre du formulaire |
| Style du borde (BorderStyle) | numéro | il détermine aspect et fonctionnalité du formulaire (dialogue) popup, redimensionnable |
| Boutons de déplacement | oui/non | pour se déplacer, en avant, en arrière, début, fin, nouveau record |
| Bouton de fermeture | oui/non | |
| Image | bitmap | fond du formulaire |
| Origine données | texte | le nom d'un table ou d'un requête ou une instruction SQL SELECT |
| Trie pour | texte | le nom d'un champ |

Ici de suite quelques événements associé au formulaires :

| événement | caractéristiques |
|---|---|
| Sur chargement (onLoad) | pour modifier des propriétés du formulaire avant qu'il soit visualisé |
| Sur ouverture (onOpen) | pour opérations précédentes la première visualisation de données |
| Sur activation (onCurrent) | avant de chaque visualisation des données |
| Sur fermeture (onClose) | pour les opérations à la fermeture du formulaire |

### 2.2.3 PERSONNALISATION D'UN FORMULAIRE

Sélectionner le formulaire et choisir Modifier, en cliquant sur l'objet il est sélectionnée et vous pouvez le déplacer et modifier sa taille. Chaque élément a des propriétés et peut réagir à des événements ; propriétés et événements dépendent du type de l'objet.

Ici une liste de quelques propriétés de la zone de texte :

| propriété | note |
|---|---|
| nom (name) | |
| visible (visibile) | |
| active (enabled) | si vrai permet d'insérer et modifier le texte |
| verruillé (looked) | si vrai permet ne permet pas de modifier le texte |
| hauteur, largeur, gauche, haut (height, width, left, top) | dimension et position sur la fenêtre |
| police, taille, … (fontname, fontsize, …) | |
| source du contrôle (controlsource) | peut être un champ d'une table ou d'une requête ou une expression |

**Figura 2-19** ***Propriétés de la zone de texte***

#### 2.2.3.1 Dimension des fenêtres

Les dimensions (hauteur et largeur), dans les fenêtres pas à l'écran plein, ils devraient être dans la proportion 1:1.6, ça correspond à *le nombre d'or*[8] à peu près 1.61803… qui vient de la proportion 1 : x = x : 1 – x.

#### 2.2.3.2 Alignement et taille des objets

- **Aligner des objets sur une fenêtre** il faut les sélectionner : clique de la souris en poussant la touche shift, un clic sur la guide numéroté en haute ou à gauche sélectionne tous les objets en vertical ou horizontal. Apres Format, Aligne ... .
- **Même taille** : sélectionner et Format, Taille ... .
- **Petites modifications de taille** : sélectionner et touche shift avec touches directionnelles.
- **Petits déplacements** : sélectionner et touche control avec touches directionnelles.

#### 2.2.3.3 Changer un texte en liste

On peut modifier le type d'objet, par exemple on peut transformer une zone de texte en zone de liste ou une liste déroulant pour choisir des données au lieu de les insérer par clavier ; ceci est surtout utilisé où les valeurs possibles sont en nombre fixe et limité, ou doivent appartenir à une autre table, par exemple un choix de zone géographique, ou le nom du matériel qui est dans la table `Magasin`.
La procédure est :

- cliquer sur l'objet avec la touche droite de la souris,
- choisir Remplacer par ...,

choisir Zone de liste déroulante ....Pour indiquer les données à choisir par la liste déroulant :

- sélectionner les propriétés du combo box,
- onglet Données,
- la propriété Origine source permet de choisir entre : Table/Requête, Liste valeurs, Liste des champs,
- dans la propriété Contenu :
  - si Table/Requête on peut choisir dans une liste de Tableaux et Requêtes ou créer une requête ; le résultat est une ou plusieurs colonnes de données, la propriété Colonne lié indique la colonne de control ; dans onglet Format la propriété Nbre colonne indique le nombre de colonnes affichées, la propriété Largeurs colonnes donne la largeur des colonnes affichées, par exemple si on veut pour choisir le nom du matériel, les propriétés vues sont semblables à :
    - Origine source : `Table/Requête`
    - Contenu : `Magasin`
    - Colonne lié : `1` c'est le CodMateriel
    - Nbre colonnes : `3` CodMateriel, Description, Quantité
    - Largeurs colonnes : `0cm;3cm;1cm`

    L'effet de `0cm;` dans Largeurs colonnes est que on verra seulement Description et quantité.
  - si Liste valeurs on insère une liste de valeurs entre apostrophes et séparé par `;` (point virgule), par exemple `'F';'M'`.
  - si Liste des champs on choisit dans une liste de Tableaux et Requêtes, le résultat sera la liste des noms des champs.

#### 2.2.3.4 Insertion de données pris d'une autre table

On peut prélever des informations appartenant à une autre table par les fonctions d'agrégation comme `RechDom`, `SomDom`, `CpteDom`[9], etc, à l'aide du Générateur d'expression (balise avec ... à droite de la proprieté Source contrôle) :

- ajouter les textes où on veut insérer les données d'une autre tableau,

[8] Le *nombre d'or*, habituellement désigné par la lettre φ(*phi*) de l'alphabet grec en l'honneur de Phidias, sculpteur et architecte grec du Parthénon.
[9] La syntaxe du Générateur d'expression diffère de celle du Basic : les nom de fonctions dépendent de la langue et les paramètres sont séparé par `;`.

- dans l'onglet Données insérer dans la propriété Source contrôle l'instructions `DLookup`, avec la syntaxe : `= RechDom ("nomchamp";"nomtable";"condition")`,
- éventuellement protéger le champ (propriétés Activé et Verrouillé).

*condition* ha la même structure de la condition de la clause `WHERE` dans les commands `SELECT` SQL.

Exemples :

- `=RechDom("[Compound]";"Farm";"[Nom]='" & [Nom] & "'")`
- `=SomDom("[Qty]";"Med";"[Nom]='" & [Nom] & "'")`

### 2.2.4 FORMULAIRE ET SUB FORMULAIRE

Pour insérer un formulaire dans un autre formulaire, par exemple dans le formulaire de la Facture le formulaire des produits vendu (relative à chaque facture) :

- Ouvrir en modification le formulaire conteneur,
- Choisir de la boite à outils la balise Sous-formulaire/Sous-Etats  ,
- Choisissez sur le formulaire où insérer le sous-formulaire ; un Assistant conduit, et si les noms des champs de liaisons sont égaux, par exemple soit *Facture* soit *Produit* contiennent le champ IDFacture, il y a presque rien à ajouter.

## 2.3 REQUÊTES

Les requêtes sont utilisées pour lier ensemble les données de deux ou plusieurs tableaux ou requêtes, par exemple la requête pour obtenir une facture utilisera les données des tableaux *Ventes*, *Clients* et *Magasin*. Cette technique exige que ces tableaux aient des champs en commune, voir le schéma de Figure 1-7 le champ *CodeClient* met en relation les tableaux Clients et Ventes et le champ *CodeMatériel* lie les tableaux Ventes et Matériels.

Pour engendrer une requête on peut choisir entre Créer une requête en mode Création o Créer une requête à l'aide de l'Assistant.. Dans le premier cas on choisi les tables et/ou les requêtes voulues. Si les champs de liaison ont le même nom dans les tableaux (comme dans l'exemple), ACCESS effectue lui-même la liaison, dans le cas contraire on doit indiquer les champs de liaisons (bouton droit de la souri pressé sur le nom du tableau que on veut lier, joindre le nom de l'autre table et relaisser la touche).

Pour indiquer les champs à insérer dans la requête : bouton gauche de la souri pressé sur le nom du champ que on veut insérer et le déplacer sur une cellule en bas.

Dans la cellule en correspondance a Tri : et au nom du champ on peut indiquer l'éventuel tri de la requête.
On peut insérer des critères de choix Dans la cellule en correspondance a Critères : et au nom du champ, par exemple si on indique `"F"` en correspondance du champ Sexe, la requête affichera seulement les ligne des femelles.

Naturellement il est possible insérer une requête comme command SQL : menu Affichage, Affichage SQL et insertion dans la zone de texte, par exemple :

```
SELECT agents.IdAgent, Agent, ventes.Montant, Ville
FROM agents INNER JOIN ventes ON agents.IdAgent = ventes.IdAgent
```

### 2.4 MODULES

Est la partie qui contient le code VBA : chaque formulaire et états peut avoir du code pour gérer des événements ; en outre variables et fonctions communes, indiqué par `Global` et `Public` respectivement, doivent être écrits dans un module dit global :

```
Global Const SortProgr = 3
Global SortAlunos
Public Function SeeTableaux(Optional Table)
    ' si le tableau existe la fonction répond avec le nom du tableau même,
    ' sinon avec la lista des tableaux separé par ;
    Dim obj As AccessObject, dbs As Object
    Set dbs = Application.CurrentData
    ' Search for objects in AllTable collection.
    Itm$ = ""
    For Each obj In dbs.AllTable
      If Left(obj.Name, 4) <> "MSys" Then
        If IsMissing(Table) Then
```

```
            Itm$ = Itm$ & ";" & obj.Name
        Else
            If Table = obj.Name Then
                SeeTableaux = Table
                Exit Function
            End If
        End If
    End If
    Next obj
    SeeTableaux = Mid$(Itm$, 2)
End Function
```

**Figure 2-20** ***Fragment de module Global***

### 2.5 MACRO

Les macros ont été la première façon de programmer d'ACCESS. Ici sont traité seulement pour la possibilité d'exécuter des automations basé sur ligne de commande.

Les macros sont une liste de commandes, choisis d'une liste déroulant. Selon la commande on peut avoir des paramètres. La commande `ExécuterCode` veut le nom d'une fonction VBA.

## 2.6 ÉTATS ET EXTRACTION DE DONNÉES

Pour engendrer un état on peut choisir entre Créer un état en mode Création o Créer un état à l'aide de l'Assistant. Ils sont des façons de travailler analogues à celle des formulaires.

Comme les formulaires, les états ont propriété et événements, en particulier l'événement Sur ouverture peut être utilisé pour changer le triage et/ou choisir entre les données.

### 2.6.1 CRÉATION ET MODIFICATION D'ÉTATS

Les États se préparent et modifient presque comme les formulaires.

Dans la création conduite il est possible grouper les données, il est un triage implicite : par exemple les ventes groupées par Ville.

#### 2.6.1.1 Ajouter des totaux

Il est possible ajouter des totaux ou des calcules à un état dans les zones En-tête de groupe et Pied de groupe pour les calcules partiels, et dans la zone Pied d'état pour les calcules globaux. Ces formules sont dans une zone de texte ou la propriété `ControlSource` contient une formule, exemple `=Somme([Montant])`.

Les zones Pied de groupe ne sont pas normalement visibles, pour les rendre visibles : dans le menu Affichage choisir Trier et grouper et choisissez Oui dans la zone Pied de groupe.

#### 2.6.1.2 Changement de page dans les groupements

Il faut indiquer dans l'Entête de groupe.

#### 2.6.1.3 Choisir les données

Dans l'événement `Sur ouverture` d'un état on peut insérer des choix sur les données en modifiante la propriété Filtre de l'état, on peut aussi la commande `DoCmd.OpenReport`, qui est engendré automatiquement par le bouton de création d'état ; cette commande prévoit un paramètre utilisé pour choisir entre les fiches, dans l'exemple il y a les instructions *VBA* pour choisir les données (en **gras** les instructions ajoutées ou modifiées).

```
Private Sub cmdEtat_Click()
On Error GoTo Err_cmdEtat_Click
    Dim stDocName As String
    where = "year(DatePrete) = " & Annee & " AND Month(DatePrete) = " & Mois
    stDocName = "etLouages"
    DoCmd.OpenReport stDocName, acPreview, , where
Exit_cmdEtat_Click:
    Exit Sub
Err_cmdEtat_Click:
    MsgBox Err.Description
```

```
    Resume Exit_cmdEtat_Click
End Sub
```

**Figure 2-21** ***Exemple d'état avec choix des données***

### 2.6.2 TRIER LES DONNÉES AVANT D'IMPRIMER

Quand on prépare un état en Mode Création, on peut aussi indiquer un triage, toutefois celui ci peut être modifié dans l'événement `Open`.Dans les états simples on doit insérer dans la propriété `OrderBy` le ou les champs de triage (même syntaxe de la clause `ORDER BY` de `SELECT`) et mettre `vrai` dans la propriété `OrderByOn` :

```
Me.OrderBy = "Nom, Qty DESC"
Me.OrderByOn = True
```

Cette technique n'est pas possible pour les états qui contiennent des groupements car celui-ci ont déjà un triage, celui du champ de groupement ; pour changer on agit sur la proprieté `ControlSource` du groupement approprié :

```
Me.GroupLevel(0).ControlSource = "Nome"    ' premier niveau de groupement
```

### 2.6.3 CRÉATION D'ÉTATS PAR VBA

ACCESS permet de créer des états au niveau de programmation, ceci est utile pour la création d'états a suite de requêtes personnalisé

```
Function CreateDynamicReport(strSQL As String, Title)
' inspiré par http://bytes.com, merci à mmccarthy, Nico5038 et FishVal
    Set dbs = Application.CurrentProject
    ' Ricerca oggetti AccessObject aperti in insieme AllReports.
    reportTemp = "rptgen"
    For Each obj In dbs.AllReports ' pour effacer l'état
        If Not obj.IsLoaded = True Then
            If reportTemp = obj.Name Then DoCmd.DeleteObject acReport, _ reportTemp
        End If
    Next obj
     intStep = 1000
     lngInitLeft = 0
     lngLeft = lngInitLeft
     lngTop = 0
     Set rpt = CreateReport 'Create the report
     ' set properties of the Report
     With rpt
         .Width = 8500
         .RecordSource = strSQL
         .Caption = Title
         .Section(acPageHeader).Height = 900
         .Section(acDetail).Height = 340
         .Toolbar = "Stampe2"          ' ajoute bar
     End With
     ' Open SQL query as a recordset
     Set db = CurrentDb
     Set rs = db.OpenRecordset(strSQL)
     ' Create Label Title
     Set lblNew = CreateReportControl(rpt.Name, acLabel, acPageHeader, , Title, 0, 0)
     lblNew.FontBold = True
     lblNew.FontSize = 14
     lblNew.SizeToFit
     ' Create labels
     For Each fld In rs.Fields
         Set lblNew = CreateReportControl(rpt.Name, acLabel, acPageHeader, _
            "", fld.Name, lngLeft, 500, intStep)
         lblNew.FontBold = True
         lblNew.FontSize = 10
         lblNew.SizeToFit
         If lngLeft <> lngInitLeft Then
            lblNew.Left = lngLeft + intStep - lblNew.Width
         End If
         lngLeft = lngLeft + intStep ' Increment top value for next control
     Next
     ' Create text box controls for each field.
```

```
    lngLeft = lngInitLeft
    For Each fld In rs.Fields
        ' Create new text box control and size to fit data.
        Set txtNew = CreateReportControl(rpt.Name, acTextBox, _
        acDetail, , fld.Name, lngLeft, lngTop, intStep)
        txtNew.FontSize = 10
        txtNew.SizeToFit
        lngLeft = lngLeft + intStep ' Increment top value for next control
    Next
    NomeReport = rpt.Name  ' parce que après ouverture disparaît le nom
    DoCmd.OpenReport rpt.Name, acViewPreview
    DoCmd.Save acReport, NomeReport
    'reset all objects
    rs.Close
    Set rs = Nothing
    Set rpt = Nothing
    Set db = Nothing
End Function
```

**Figure 2-22** ***Création d'état***

## 2.7 EXPORTATION DE DONNÉES

L'instruction pour exporter des données est `DoCmd.OutputTo` ; `DoCmd.OutputTo` peut préparer les donnés avec différents formats comme une page HTML à partir d'un formulaire ou une feuille Excel avec les données d'une table ou d'une requête, par exemple :

```
DoCmd.OutputTo acOutputQuery, "rqtStatistiques", acFormatXLS, xlsDocName
Call Shell("Excel " & xlsDocName, 1)
```

La première instruction prépare un fichier Excel à partir d'une requête, la deuxième instruction ouvre Excel avec ce fichier.

## 2.8 QUELQUE FONCTION DE VBA

Outre aux instructions du langage sont accessibles des librairies pour ajouter des fonctionnalités, voir dans le menu de VB Outils, Références.

### 2.8.1 ACCÉDER AUX CHAMPS D'UNE TABLE

Pour accéder à un champ d'une fiche, on utilise la fonction `DLookup` :

```
champ = DLookup("NomDuChamp","NomDeTable","Critères")
```

Ici dans l'exemple on voit comme prélever plus d'un champ (de la même fiche).

```
ClTrnProm = DLookup("Classe & ',' & Turno & ',' & Promovido", "AnosAlunos", _
              "AnoAluno ='" & tAnoEscolar & "' AND IDAluno =" & Me.[Progr])
If Nz(ClTrnProm, "") <> "" Then
   Fields = Split(ClTrnProm, ",")    ' Separe les champs
   Classe = IIf(Fields(2) = "Sim", Val(Fields(0)) + 1, Fields(0))
```

Il y a aussi autres fonctions comme : `Dsum`, `Dcount`, `Dmin` et `Dmax`.

### 2.8.2 CONTRÔLES PERSONNALISÉS

Il est possible associer à un événement des élaborations en Visual Basics. En sélectionnant, dans les propriétés, l'événement, paraît à droite un bouton, qu'il propose une liste de laquelle faut choisir Générateur de code, tel choix crée en Visual Basics le squelette d'une procédure. Ici de suite un exemple de code activé quand on sort du champ de nom `tst` et qui transforme le texte en minuscule sauf la première lettre.

```
tst = UCase(Left(tst, 1)) & LCase(Mid(tst, 2))
```

### 2.8.3 CONTRÔLES SUR UN CHAMPS

Dans les événements Sur sortie (OnExit) et Sur perte focus (OnLostFocus) on peut contrôler ou modifier le donné inséré ; la différence entre les deux événements est que dans Sur sortie on peut indiquer de rester avec le curseur sur la zone de texte (voir exemple).

```
Private Sub Sexe_Exit(Cancel As Integer)
Sexe.Value = UCase(Sexe.Value)
If InStr("MF", Sexe.Value) = 0 Then
    MsgBox "Sexe interdit!"
    Cancel = True
End If
End Sub
```

### 2.8.4 OPÉRATIONS PRÉ-AFFICHAGE DE DONNÉES

Dans les événements du formulaire Sur Affichage (OnCurrent) on peut effectuer des opérations ; dans l'exemple pour les lignes affichées on visualise les valeurs de Prix et Stock prélevé du tableau Magasin.

```
Private Sub Form_Current()
Prix = 0
Stock = 0
If Not Me.NewRecord Then
    Prix = DLookup("Prix", "Magasin", "CodeMateriel =" & CodeM)
    Stock = DLookup("Stock", "Magasin", "CodeMateriel =" & CodeM)
End If
End Sub
```

### 2.8.5 EXAMEN DES OBJETS D'UN FORMULAIRE

L'exemple montre comme accéder aux objets d'un formulaire, dans l'exemple le contenu des étiquettes est changé (pour changer la langue).

```
Public Sub ImpostaLingua(Frm)
' internazionalizzazione delle Forms presuppone label in Italiano
If Not IsEmpty(Lingua) And Lingua <> "Italiano" Then
     For Each MyCtrl In Frm.Controls   ' Esegue un'iterazione in ogni elemento
       If MyCtrl.ControlType = acLabel Then
         Label = Nz(DLookup(Lingua, "Dizionario", "Italiano = '" &  _
                 MyCtrl.Caption & "'"))
         If Label <> "" Then MyCtrl.Caption = Label
       End If
     Next
End If
End Sub
...
ImpostaLingua Me
...
```

### 2.8.6 CRÉATION D'UN OBJET

On peut créer des objets comme requêtes et reports, il faut que soit accessible la librairie DAO 3.6 Object Library[10] (menu de VB Outils, Références) . L'exemple crée une requête pour accéder aux données.

```
Public Sub CreaQuery(strQuery, Sql)
' parameters : nom et command SQL
On Error Resume Next
Set dbs = CurrentDb
dbs.QueryDefs.Delete strQuery
On Error GoTo Err_CreaQuery
Set qdf = dbs.CreateQueryDef(strQuery, Sql)
Exit_CreaQuery:
  Set qdf = Nothing
  Set dbs = Nothing
Exit Sub
Err_CreaQuery:
  MsgBox Err.Description
  Resume Exit_CreaQuery
End Sub
```

## 2.9 LIGNE GUIDE POUR BÂTIR UNE APPLICATION

Ce paragraphe indique des techniques pour bâtir une application, en particulier seront traité :

- conseil sur le nom des objets,
- accéder aux différentes fonctionnalités de l'application (panneau de contrôle),
- rendre l'application utilisable par différents sujets (personnalisation),
- séparer les données de l'application,
- utiliser des sources de données préexistantes,
- protéger et contrôler les données.

[10] DAO (*Data Access Objects*) est une interface de programmation pour accéder à des Base de Données. Cette librairie ne sera pas disponible dans les Systèmes d'exploitation à 64 bits.

### 2.9.1 CONSEIL SUR LE NOM DES OBJETS

On peut utiliser quelconque nom pour les objets, toutefois il faudrait utiliser toujours des nom significatifs, surtout pour les objets auxquels on ajoute le traitement des événements. La figure qui suive utilise, partiellement, la nomenclature dite *Hongrois* : des lettres minuscules initiales qui déterminent l'objet, suivi par le nom de l'objet dont le premier caractère est majuscule :

| Nom Objet | Type | Note |
|---|---|---|
| `Ventes` | Table | Remarquez le pluriel |
| `frmVentes` | Formulaire | Formulaire relatif aux ventes |
| `rqtVentesClients` | Requête | Requête relatif aux ventes des Clients |
| `eVentesClients` | Etat | Etat relatif aux ventes des Clients |
| `cmdVentes` | Bouton | Bouton qui démarre le formulaire ou l'état des Ventes |
| `etqNouveau` | Etiquette | |

**Figure 2-23 *Nommes des objets***

Vous pouvez faire exception pour les Tables et les noms des Zones de texte, surtout car ACCESS donne aux Zones de texte relatives à champs de table et requêtes, le nom du champ même, par exemple si la table Clients contient le champ `Adresse`, sur le formulaire obtenu par l'Assistant, le nom de l'objet zone de texte correspondant sera `Adresse`.

Il est important utiliser le même nom (et même taille de donnée) pour les champs utilisé pour lier ensemble les table, par exemple si la table `Ventes` contient `CodeClient` et `CodeMateriel` (format numérique Entier long), et les tableaux `Clients` et `Magasin` contiennent respectivement `CodeClient` et `CodeMateriel`, ACCESS sera capable de faire les liaisons automatiquement pour vous.

### 2.9.2 LE FORMULAIRE DE CONTRÔLE

Le *formulaire de contrôle* est un formulaire qui contient les boutons pour démarrer les principales fonctions de l'application, par exemple :

- accéder aux formulaires de gestion des données,
- production d'états,
- fonction de recherche et d'extraction des données,
- élaborations techniques comme le sauvetage de la base des données et la modification des personnalisations,
- terminer l'application.

Le formulaire doit être créé avec la fonction Créer un formulaire en mode Création.
Du panneau des instruments sélectionner les boutons et avec la souris portez-le sur le formulaire. Pour les principales opérations ACCESS construit automatiquement le code Visual Basic.

Pour qu'ACCESS au commencement présente le "formulaire de contrôle", il faut accéder à la voix de menu Outils, choisir Démarrage.... et choisir le nom du panneau de démarrage du combo box Afficher formulaire/page.

Pour terminer l'application à l'aide d'un bouton, dans l'événement Sur click insérez la méthode `DoCmd.Quit`.

### 2.9.3 PERSONNALISATION

Une application peut être utilisée par différents utilisateurs, dans ce cas il est important déterminer les informations variables, en général sont des informations comme le nom de la Société, l'adresse etc. Ces données doivent être mémorisé dans une table.

#### 2.9.3.1 Menu personnalisé

Si on élimine les menus d'ACCESS est nécessaire ajouter certaines fonctionnalités avec ou des menus ou des menus contextuels (évocables par la touche droite de la souris) :

- Affichage, Barre d'outils, Personnaliser,
- dans l'onglet Barres d'outils, choisir Nouveau et donnée un nom, fermez par OK.

- dans l'onglet Barres d'outils, ayant sélectionnée le nouveau menu, cliquez sur Propriétés, choisissez de la liste Type le type du menu :
  - Barre menu
  - Barres d'outils
  - FenIndépendante (Pop Up)
- Choisissant Barre menu et Barres d'outils appaire une fenêtre dans laquelle on peut faire glisser les fonctions voulues :
  - onglet Commands
  - touche droit de la souris sur l'icône présent dans la zone Commands
  - déplacer dans la fenêtre
- Insérez le menu dans le formulaire ou état voulu :
  - Proprieté onglet Autres, MenuBar ou ToolBar ou ShortCutMenuBar

    L'insertion du Menù peut aussi etre faite par VBA :

```
Set rpt = CreateReport
With rpt     ' set properties of the Report
    .Width = 8000
    ...
    .Toolbar = "Etats"   ' insère la barre d'outils Etats    '
    ...
```

**Attention**, pour les Fenêtres Indépendantes il faut partir par une Barre d'outils et la transformer quand elle est complète. Pendant que les Barres des Menus et Barres d'Outils sont listées dans l'onglet Bar d'outils et dans la liste de la fenêtre des propriétés, les Fenêtres Indépendantes sont pressentes seulement dan cette dernière liste ; en autre mot, pour modifier les fonctions d'une Fenêtre Indépendante, il faut la transformer en Barre d'Outils et après la retransformer en Fenêtre Indépendante.

### 2.9.4 MULTI-UTILISATION

Les Base de Données ACCESS peuvent être utilisé en contemporanéité par plusieurs utilisateurs ceci signifie que la Base de Données est hébergé avec éventuellement la Base de Données avec l'application, sur un ordinateur en réseau. On estime que les applications MS ACCESS ne devraient pas dépasser 25 utilisateurs actives simultanément.

### 2.9.5 PROTECTION ET SÛRETÉ

L'aide en ligne d'ACCESS et la documentation que on peut trouver sur INTERNET expliquent plusieurs techniques pour la protection ; ici sans vouloir être exhaustive, il y a quelque indications.

#### 2.9.5.1 Utilisation de l'application

Pour permettre l'utilisation de l'application seulement aux personnes autorisées on peut :

- insérer la requête de nom d'utilisateur et de mot de pas, ceci on le doit gérer avec un formulaire et une table qui contienne les informations de l'utilisateur. Cette technique est faible parce que on peut voir le contenu des tableaux.
- Utiliser les protections de Windows, qui sont accessibles par le menu Outils et dont la plus simple est l'utilisation d'un mot de pas.

#### 2.9.5.2 Protection du code VBA

- menu Outils,
- choisir Utilitaires de base de données,
- choisir Créer un fichier MDE…,
- confirmer ou changer le nom proposé,
- cliquez sur le bouton Enregistrer.

Cette opération engendre l'application sous forme de fichier `.mde`. qui sera distribué à l'utilisateur final au lieu de l'application `.mdb`.

### 2.9.5.3 Protection contre modifications accidentelles

Pour protéger l'application contre des modifications accidentelles ou voulues, on doit éviter que l'utilisateur puisse entrer en *Mode projet*, ceci on l'obtienne dans le menu Outils, Démarrage.... Ici on peut décocher les choix relatives aux menus pour les rendre indisponibles au démarrage, compris la visualisation de la fenêtre de la base de données. En outre l'exécution du command VBA : `CommandBars("Menu Bar").Enabled = False` élimine tous les menus.

S'il y a la nécessité d'accéder pour modifier l'application, on doit démarrer avec la touche shift pressé.

A ce niveau la protection est faible, il faut interdire la touche ajoutant la propriété "AllowBypassKey" à l'ensembles *Properties* de l'objet *Database*, cette technique devrait être renforcé avec du code pour inhiber cette touche et pour permettre la réhabilitation.

L'exemple suivant explique une possible solution :

**Interdiction**

Au debout de l'application, par exemple dans l'avènement de chargement `Load` du le formulaire initial, on doit donner la valeur `False` à la propriété `AllowByPassKey` ; ici on utilise une fonction (`ChangeProperty`) dans laquelle la propriété, s'elle n'est pas présent, elle est engendré :

```
CommandBars("Menu Bar").Enabled = False
ChangeProperty "AllowBypassKey", False
```

```
Public Function ChangeProperty(strPropName As String, varPropValue) As Integer
    ' cette fonction modifie la valeur d'une propriété
    ' Si la propriété n'existe pas, elle est créé
    Dim dbs As Object, prp As Variant
    Const conPropNotFoundError = 3270
    Set dbs = CurrentDb
    On Error GoTo Change_Err
    dbs.Properties(strPropName) = varPropValue
    ChangeProperty = True
Change_Bye:
    Exit Function
Change_Err:
    If Err = conPropNotFoundError Then    ' propriété pas trouvé
        Set prp = dbs.CreateProperty(strPropName, 1, varPropValue)
        dbs.Properties.Append prp
        Resume Next
    Else
        ' Erreur inconnu
        ChangeProperty = False
        Resume Change_Bye
    End If
End Function
```

**Figure 2-24** ***Fonction pour gérer la touche shift***

**Réhabilitation**

Insérez cette commande dans l'événement `MouseDown` d'une image, ou d'une étiquette :

```
If Button = acRightButton Then ChangeProperty "AllowByPassKey", True
```

(Pour le rendre plus sur on peut aussi demander un mot de passe).

Pour pouvoir accéder à l'application pas protégé il faut :

- démarrer l'application,
- presser le bouton droit sur l'image ou l'étiquette choisie,
- fermer l'application,
- démarrer l'application avec la touche shift pressé.

Un démarrage sans touche shift pressé rétablisse les condition de sûreté.

### 2.9.5.4 Sûreté des données

Au-delà de la protection de l'application contre des accès pas autorisés, il faut prévoir la sauvegarde périodique des données, par

```
echo off
REM où sont les données
D:
cd D:\CONDOR\SERMIG\FARO
if exist faro3.mdb del faro3.mdb
if exist faro2.mdb ren faro2.mdb faro3.mdb
if exist faro1.mdb ren faro1.mdb faro2.mdb
copy faro.mdb faro1.mdb
pause Presser une touche pour continuer
exit
```

**Figure 2-25** ***Procédure de sauvegarde***

exemple à chaque fin de l'application et/ou chaque semaine. La sauvegarde devrait être fait sur une unité de mémorisation différente de celle de travail (autre disque dur, pen drive, CD, etc.). Il faut prévoir dans la table de personnalisation la mémorisation de la date de la dernière sauvegarde et le control de cette au démarre de l'application.

La sauvegarde sur le même disque est tolérable seulement pour les sauvegardes en fin de travail.

Ici à droite une procédure de sauvegarde sous forme de procédure DOS `.bat`, il est prévu trois générations de la même Base de Données ; pour exécuter cette procédure ajouter une instruction VBA comme la suivante :

```
Call Shell("D:\CONDOR\SERMIG\FARO\sauve.bat", vbNormalFocus)
```

#### 2.9.5.5 Sûreté de l'application

La sûreté de l'application doit être géré à niveau de programmation, ici quelque point :

- demander la confirmation des opérations pas recouvrables, comme l'effacement de données.
- Désactivez ou cachez les commandes pas pertinents.

### 2.9.6 COMPRESSION DE LA BASE DE DONNÉES

Les Base de Données ACCESS, même dans les versions plus récentes (ACCESS 2007), pendant l'utilisation augmentent l'espace mort, pour cette raison il faut prévoir de temps en temps, de compacter la Base de Données, par exemple on peut ajouter aux commande de la :

```
msaccess faro.mdb /compact
```

### 2.9.7 SÉPARATION ENTRE DONNÉES ET APPLICATION

Presque toutes applications ont besoin de modification : pour améliorer, pour introduire des nouvelles fonctions ou modifier et enfin aussi pou corriger des fautes.

Avec une Base de Données qui contient données et application, modifier celle-ci peut provoquer des dommages aux données. Pour éviter ceci ACCESS permet de séparer les deux composants, c'est-à-dire que l'application sera composé par deux (ou plus) Base de Données ACCESS, l'une avec les données et l'autre avec l'application (formulaires, requêtes, états et code VBA). Cette dernière au lieu des tableaux, aura des liaisons à les tables présentes dans l'autre.

La procédure pour obtenir la liaison est la suivante :

- ouvrir ACCESS applications et dans la gestion des Table
- choisir Nouveau
- Importer la table
- chercher le fichier ACCESS qui contient les données,
- choisir toutes les tableaux dont on veut la liaison,
- cliquer sur le bouton OK.

Mais il y a une raison plus importante pour séparer données et application : les données sont propriété et responsabilité de l'utilisateur, au contraire l'application est responsabilité du développeur.

#### 2.9.7.1 Changement de répertoire des données

La séparation entre données et logiciel crée une liaison "fixe" entre les deux Bases de Données, pour changer ça

#### 2.9.7.2 Mise à jour sur la Base de Données du propriétaire

La nécessité de modifier la Base de Données qui contient les Données comporte des activités délicates. Une possible ligne d'action est de,

L'exemple suivante, qui utilise la librairie DAO, crée une table sur une autre Base de Données, insère une ligne sur la table et engendre la liaison.

```
Private Sub addTable(DBName As String, tbName As String)
Dim db As Database
Dim tdf As TableDef
Set db = DBEngine.OpenDatabase(DBName)
db.Execute "CREATE TABLE " & tbName & " (Profissao VARCHAR(30))"
db.Execute "INSERT INTO " & tbName & " VALUES ('Ferreiro')"
Set tdf = db.CreateTableDef(tbName)
tdf.Connect = ";DATABASE=" & DBName
```

```
tdf.SourceTableName = tbName
CurrentDb.TableDefs.Append tdf
db.Close
Set db = Nothing
End Sub
```

### 2.9.8 UTILISER DES SOURCES DE DONNÉES PRÉEXISTANTES

#### 2.9.8.1 D'EXCEL à ACCESS

Une feuille D'EXCEl qui contienne des données structuré peut-être transformé aisément en une table d'ACCCES, pourvu que la première ligne contienne les entêtes des colonnes (qui seront transformé en noms des champs) :

- choisir Fichier,
- Données externes,
- Importer …,
- Dans Type de fichier : choisissez Microsoft Excel (*.xls)
- Choisissez le fichier EXCEL et pousser le bouton Importer
- Apparaître l'Assistant Importation de feuille de calcul qui vous permet d'importer les données.

Exemple :

| Nom | Adresse | Ville | Dette | DMaJ |
|---|---|---|---|---|
| Ector Coulibaly | Rue Bamako | Kati | 50000 | 08/10/2006 |
| Jean Ross | Rue Oriani | Turin | 150000 | 03/10/2006 |
| Amina Toure | Rue Bamako | Bamako | 25000 | 08/09/2006 |
| Fatuma Diallo | Rue 99 Koko | Kati | 50000 | 08/10/2006 |

**Figure 2-26** ***Conversion d'Excel à ACCESS***

**Attention** : peut être que on doit modifier la dimension les champs reconnu comme texte parce que ils sont engendré avec une dimension de 255 caractères.

# 3. INDEXES

## 3.1 INDEX ANALYTIQUE

## 3.2 INDEX DES FIGURES