SQL

Jusqu'à présent, nous avons vu comment créer une requête en utilisant la fenêtre requête d'ACCESS. Il existe une autre façon de faire les requêtes : en passant directement par le langage SQL. D'ailleurs, ACCESS utilise aussi ce langage pour faire ses requêtes, il traduit ce que vous avez entré en SQL, puis exécute la requête SQL.

1. Qu'est-ce que SQL ?

SQL est un langage de manipulation de bases de données mis au point dans les années 70 par IBM. Il permet, pour résumer, trois types de manipulations sur les bases de données :

?? La maintenance des tables : création, suppression, modification de la structure des tables.
?? Les manipulations des bases de données : Sélection, modification, suppression d'enregistrements.
?? La gestion des droits d'accès aux tables : Contrôle des données : droits d'accès, validation des modifications.

L'intérêt de SQL est que c'est un langage de manipulation de bases de données standard, vous pourrez l'utiliser sur n'importe quelle base de données, même si, à priori, vous ne connaissez pas son utilisation. Ainsi, avec SQL, vous pouvez gérer une base de données Access, mais aussi Paradox, dBase, SQL Server, Oracle ou Informix par exemple (les bases de données les plus utilisées).

Attention : Le langage SQL a souvent été implémenté de façon différente. Les commandes de base sont toujours les mêmes mais ont parfois des variantes ou des extensions. La version de SQL implémentée dans Access peut parfois être un peu différente de la version ANSI qui est la version standardisée.

2. La maintenance des bases de données

La première série de commandes sert à la maintenance de la base de données : création des tables et des indexes, modification de la structure d'une table ou suppression d'une table ou d'un index.

Attention : La maintenance des bases de données dépend étroitement de la base de données utilisée, notamment en ce qui concerne les types de données, vérifiez donc toujours les types de données supportés par votre SGBD avant de créer une table avec SQL. Il se peut qu'il en supporte plus que ceux indiqués plus bas.

SQL dispose pour cela des instructions suivantes :

?? `ALTER TABLE`
?? `CREATE TABLE`
?? `CREATE INDEX`

2.1 La commande CREATE TABLE

La commande CREATE TABLE permet de créer une table dans la base de données courante.
Sa syntaxe est la suivante :

```
CREATE TABLE    table
                (champ type CONSTRAINT champ propriétés, ... );
```

Paramètre	Signification
Champ	Nom du champ
Type	Type de données, dans la plupart des versions de SQL, vous aurez droit aux types de données suivants : ?? `Char(x)` : chaîne de caractères, x est le nombre maximum de caractères autorisés dans le champ. ?? `Integer` : Nombre entier, positif ou négatif ?? `Decimal (x,y)` : Nombre décimal, x est le nombre maximum de chiffres et y le nombre maximum de chiffres après la virgule. *Decimal ne fonctionne pas avec Access, il ne supporte que le type 'float' (flottant), le type float ne permet pas d'indiquer le nombre de chiffres après ou avant la virgule* ?? `Date` : Une date et/ou heure ?? `Logical` – Deux valeurs possibles : oui / non
propriétés	Propriétés du champ : ?? `NULL ou NOT NULL` : autorise ou non que le champ puisse être vide. ?? `UNIQUE` : indique que deux enregistrements ne pourront avoir la même valeur dans ce champ. ?? `PRIMARY KEY` : indique que ce champ est la clef primaire ?? `CHECK (condition)` : équivaut à la propriété "ValideSi" d'Access, va forcer SQL a faire une vérification de la condition avant de valider la saisie, exemple : CHECK (prix > 100) interdira la saisie dans ce champ si la valeur contenue dans le champ prix est inférieure à 100. *CHECK ne fonctionne pas avec Access*. ?? `DEFAULT = valeur`, place une valeur par défaut dans le champ (ne fonctionne pas avec Access) *DEFAULT ne fonctionne pas avec Access*

Exemple : Créer la nouvelle table "table_test" contenant deux champs : un champ avec un entier qui doit toujours être saisi et un champ contenant une chaîne de 5 caractères :

```
CREATE TABLE table_test (champ1 integer CONSTRAINT champ1 NOT
NULL, champ2 char(5));
```

2.2 La commande ALTER TABLE

La commande ALTER TABLE permet de modifier la structure d'une table, sa syntaxe est la suivante :

2.2.1 ALTER TABLE table

```
Action (spécifications du champ);
```

ALTER TABLE permet trois actions, ces actions sont :

- **ADD** Ajoute un champ a une table
- **DROP** Supprime un champ d'une table
- **MODIFY** Modifie les caractéristiques d'un champ

Après l'action, on indique, entre parenthèses, les spécifications du champ de la même façon que pour la commande CREATE TABLE. On ne peut faire qu'une action à la fois (ajout, suppression ou modification dans la même commande).

Exemple :

Ajout d'un champ Date :

```
ALTER TABLE table_test  ADD champ3 Date;
```

Suppression du champ2 :

```
ALTER TABLE table_test DROP champ2;
```

2.3 La commande CREATE INDEX

La commande CREATE INDEX permet de créer un index sur une table, sa syntaxe est :

```
CREATE UNIQUE INDEX nom_index
     ON table (liste de champs);
```

Pour vous souvenir ce qu'est un index, retournez voir le chapitre 1.

Nom_index est le nom de l'index, table est le nom de la table.

Nous avons vu dans le chapitre 1 qu'un index peut être composé d'un ou de plusieurs champs, la liste de ces champs est indiquée entre parenthèses après le nom de la table.
Enfin, la clause UNIQUE indique à SQL si l'index créé va être unique ou s'il peut contenir plusieurs fois la même valeur dans la table, s'il peut contenir plusieurs fois la même valeur, il ne faudra pas ajouter la clause UNIQUE.

Exemple : création d'un index sur le champ1

```
CREATE UNIQUE INDEX index1 ON table_test (champ1);
```

3. Les manipulations des bases de données

Une fois les tables créées, on peut commencer à y insérer des données, les mettre à jour, les supprimer ou y faire des requêtes. Toutes ces opérations sont des opérations de manipulation des bases de données.

Pour effectuer ces manipulations, SQL dispose de 5 instructions :

?? INSERT
?? UPDATE
?? DELETE
?? SELECT
?? CREATE VIEW (non utilisé dans Access)

3.1 La commande INSERT

3.1.1 Présentation & syntaxe

La commande INSERT est utilisée pour ajouter des enregistrements ou des parties d'enregistrements dans des tables. Elle est utilisée généralement sous deux formes :

1ère forme

```
INSERT
INTO      table (champ1,champ2,...)
VALUES    ('valeur1','valeur2',...);
```

Cette forme est utilisée lorsqu'on veut insérer un seul enregistrement ou une partie d'un seul enregistrement. On créera un nouvel enregistrement dont le contenu du champ1 sera valeur1, le contenu du champ2 sera valeur2, etc...

2ème forme

```
INSERT
INTO      table (champ1,champ2,...)
          (requête);
```

Dans cette seconde forme, le résultat de la requête va être inséré dans les champs indiqués de la table. Cette méthode est utilisée lorsque plusieurs enregistrements sont ajoutés simultanément.

Dans les deux cas, les valeurs insérées doivent correspondre au type de données du champ dans lequel l'insertion va être faite, on ne peut pas, par exemple demander l'insertion d'une chaîne de caractères dans un champ de type numérique ou monétaire. Les chaînes de caractères doivent être placées entre apostrophes ('), les champs numériques ou vides (NULL) ne doivent pas être placés entre apostrophes.

3.1.2 Insérer tout ou une partie d'un enregistrement

Si des valeurs doivent être insérées dans tous les champs de l'enregistrement de la table, la liste des noms des champs n'a pas besoin d'être explicitement indiquée dans la commande. Les valeurs des champs à insérer doivent cependant apparaître dans le même ordre que les noms des champs lors de la création de la table, sans oublier un seul champ.
Faites attention en utilisant cette syntaxe, si la structure de la table change plus tard, la commande qui était bonne risque de ne plus fonctionner correctement. Il vaut mieux toujours indiquer explicitement le nom des champs sur lesquels on veut agir.

La syntaxe est alors :

```
INSERT
INTO      table
VALUES    ('valeur1','valeur2','valeur3',...);
```

Par exemple, dans notre table Client, si nous voulons insérer un nouveau client, nous allons entrer :

```
INSERT
INTO Clients
VALUES (100,'Mr','Dupond','Jean','rue de la paix',’75000’,
,'Paris',NULL);
```

La commande ci-dessus va insérer un enregistrement dans la table 'clients' avec les informations suivantes : Le client 100, dont le titre est 'M.', dont le nom est 'Dupond', dont le prénom est 'Jean', l'adresse est 'Rue de la paix', le code postal est 75000 et la ville est 'Paris'.

Notez que le numéro de client, qui est un champ de type 'NuméroAuto' a été indiqué explicitement. Lors d'une création d'un enregistrement avec une commande SQL, le numéro automatique n'est pas toujours généré automatiquement, vérifiez les possibilités offertes par votre SGBD.

Notez aussi l'utilisation de NULL : Si on insère moins de valeurs de champ qu'il y a de champs dans la table, soit on écrit explicitement le nom des champs à insérer, soit on insère la valeur NULL dans le champ où il n'y a rien à mettre, à condition que la propriété du champ "NULL interdit" soit à NON, sinon, il y aura une erreur lors de la création de l'enregistrement.

La syntaxe est :
```
INSERT
INTO table (champ1, champ3)
VALUES ('valeur1','valeur 3');
```

Ou
```
INSERT
INTO table
VALUES ('valeur1',NULL,'valeur3');
```

Dans notre exemple, le champ 'Observations' à sa propriété "NULL interdit" à NON, on n'est donc pas obligé de saisir des observations, ce qui est le cas ici, on entre donc NULL, ce qui signifie à Access que le champ est vide.

3.1.3 Insérer plusieurs enregistrements ou plusieurs parties d'enregistrements

On peut insérer simultanément plusieurs enregistrements ou parties d'enregistrements dans une table. Dans ce cas, les données insérées vont être récupérées dans une ou plusieurs autres tables.
Par exemple, si nous voulons placer dans une table nommée "Clients_stq" les clients habitant Saint-Quentin, nous allons procéder ainsi :

1) Créer une table "Clients_stq" avec Access, nous allons supposer que cette table comporte 3 champs: le nom, le prénom et l'adresse.

2) Taper la commande :

```
INSERT
INTO     Clients_stq(nom,prenom,adresse)
         (SELECT nom,prenom,adresse
         FROM Clients
         WHERE ville='Saint-Quentin')
```

La partie entre ( ) est une requête SQL, nous verrons cela en détail plus bas, elle va sélectionner les champs nom, prénom et adresse de la table Client pour les enregistrements dont la ville est "Saint-Quentin". Le résultat de cette requête va être inséré dans la table Clients_stq.

*La commande **INSERT INTO** avec requête ne déplace pas les enregistrements de la table "Clients" vers la table "Clients_stq", elle se contente de faire une copie. Pour effacer les enregistrements de la table Clients, nous utiliserons la commande DELETE.*

3.2 La commande UPDATE

La commande UPDATE est utilisée pour changer des valeurs dans des champs d'une table.
Sa syntaxe est :

```
UPDATE     table
SET        champ1 = nouvelle_valeur1,
           champ2 = nouvelle_valeur2,
           champ3 = nouvelle_valeur3
WHERE condition;
```

La clause SET indique quels champs de la table vont être mis à jour et avec quelles valeurs ils vont l'être. Les champs non spécifiés après la clause SET ne seront pas modifiés.

Par exemple, si nous voulons, dans la table produit, modifier le prix d'un produit dont le nom est "prod1", nous taperons :

```
UPDATE produits
SET prix_unitaire = 1000
WHERE libelle = 'prod1';
```

La commande UPDATE affecte tous les enregistrements qui répondent à la condition donnée dans la clause WHERE. Si la clause WHERE est absente, tous les enregistrements de la table seront affectés.

Par exemple, si nous tapons :

```
UPDATE produits
SET prix_unitaire = 1000;
```

Le prix unitaire de TOUS les produits de la table produit va être modifié.

Tout comme la commande INSERT, la commande UPDATE peut contenir une requête. Dans ce cas la syntaxe est la suivante :

```
UPDATE      table
SET         champ1 = nouvelle_valeur1,
            champ2 = nouvelle_valeur2,
            champ3 = nouvelle_valeur3
WHERE       condition =
            (requête);
```

La requête est une requête faite avec la commande SELECT.

3.3 La commande DELETE

Pour supprimer des enregistrements d'une table, utilisez la commande DELETE. La syntaxe est la suivante :

```
DELETE
FROM        table
WHERE       condition;
```

On ne peut pas supprimer seulement le contenu de quelques champs des enregistrements. La commande DELETE supprime des enregistrements entiers, c'est pour cela qu'il n'est pas nécessaire d'indiquer ici des noms de champs. La condition spécifiée après WHERE va déterminer quels sont les enregistrements à supprimer.

Par exemple, pour supprimer tous les clients dont la ville est Saint-Quentin :

```
DELETE
FROM Clients
WHERE ville='Saint-Quentin';
```

Pour supprimer tous les enregistrements d'une table, n'indiquez pas de clause WHERE :

```
DELETE FROM table;
```

Cette variante de la commande DELETE ne supprime pas la table, elle supprime seulement les enregistrements contenus dans cette table et laisse une table vide.

On peut aussi, comme précédemment utiliser une requête qui servira à déterminer la condition de la suppression. La syntaxe est la suivante :

```
DELETE
FROM        table
WHERE       condition =
            ( requête );
```

3.4 La commande SELECT

3.4.1 Présentation & Syntaxe

La commande SELECT est la commande la plus complexe de SQL. Cette commande va servir à faire des requêtes pour récupérer des données dans les tables. Elle peut être associée à une des commandes de manipulation de tables vues avant pour spécifier une condition.

Sa syntaxe est :

```
SELECT     champ1, champ2, champ3, ...
FROM       table;
```

S'il y a plus d'un champ spécifié après SELECT, les champs doivent être séparés par des virgules.
Les champs sont retournés dans l'ordre spécifié après la clause SELECT, et non pas dans l'ordre qu'ils ont été créés dans la table.

Par exemple :

Pour sélectionner les champs "prénom" et "nom" de tous les enregistrements de la table Clients :

```
SELECT prénom,nom
FROM clients
```

Va renvoyer les prénom et nom de tous les clients de la table Clients.

Si on veut récupérer tous les champs des enregistrements sélectionnés, la syntaxe est la suivante :

```
SELECT     *
FROM       table;
```

Les clauses SELECT et FROM doivent obligatoirement apparaître au début de chaque requête, on peut, ensuite, indiquer des critères de sélection avec la clause WHERE :

```
SELECT     *
FROM       table
WHERE      condition;
```

Par exemple, pour sélectionner tous les Clients de la table "Clients" dont le code postal est 75000 :

```
SELECT *
FROM Clients
WHERE code_postal = 75000;
```

3.4.2 Les opérateurs de condition

On peut utiliser les opérateurs suivants dans les conditions :

Opérateur	Signification
=	Egal
<>	Différent (parfois noté aussi !=)
<	Inférieur
>	Supérieur
<=	Inférieur ou égal
>=	Supérieur ou égal

Pour sélectionner tous les articles dont le prix est supérieur à 100 F :

```
SELECT *
FROM Clients
WHERE prix_unitaire > 100;
```

3.4.3 Opérateurs logiques

Il est possible de combiner plusieurs conditions avec des opérateurs logiques :

L'opérateur **AND** réunit deux ou plusieurs conditions et sélectionne un enregistrement seulement si cet enregistrement satisfait TOUTES les conditions listées. (C'est-à-dire que toutes les conditions séparées par AND sont vraies). Par exemple, pour sélectionner tous les clients nommés 'Dupond' qui habitent Saint-Quentin :

```
SELECT *
FROM Clients
WHERE nom = 'Dupond' AND ville = 'Saint-Quentin';
```

L'opérateur **OR** réunit deux conditions mais sélectionne un enregistrement si UNE des conditions listées est satisfaite. Par exemple, pour sélectionner tous les clients nommés 'Dupond' ou 'Durant' :

```
SELECT *
FROM Clients
WHERE nom = 'Dupond' OR nom = 'Durant';
```

AND et **OR** peuvent être combinés :

```
SELECT *
FROM Clients
WHERE nom = 'Dupond' AND (ville = 'Saint-Quentin' OR ville =
'Paris');
```

Nous sélectionnons ici les clients nommés "Dupond" qui habitent soit à Saint-Quentin, soit à Paris.
Pourquoi avons-nous placé des parenthèses ? Pour résumer, on peut dire que l'opérateur AND a une plus grande priorité que l'opérateur OR. Ce qui signifie que SQL va d'abord sélectionner les conditions séparées par des AND, puis celles séparées par des OR, si on avait omis les parenthèses ici, SQL aurait cherché les clients nommés "Dupond" vivant à Saint-Quentin ou les clients habitant à Paris, ce qui n'est pas le but recherché. Pour généraliser, mettez toujours des parenthèses pour bien séparer vos conditions.

3.4.4 Clauses IN et BETWEEN

Pour sélectionner des enregistrements dont la valeur d'un champ peut être comprise dans une liste ou entre deux valeurs, on utilise les clauses IN et BETWEEN.

Par exemple : Pour sélectionner les clients vivant à Saint-Quentin ou Paris :

```
SELECT *
FROM Clients
WHERE ville IN ('Saint-Quentin', 'Paris');
```

Ou pour sélectionner les produits dont le prix est compris entre 100 et 1000 F :

```
SELECT *
FROM Produits
WHERE prix_unitaire BETWEEN 100 AND 1000;
```

Pour sélectionner les produits dont le prix n'est pas dans cet intervalle :

```
SELECT *
FROM Produits
WHERE prix_unitaire NOT BETWEEN 100 AND 1000;
```

De la même façon, NOT IN sélectionne les enregistrements exclus de la liste spécifiée après IN.

Notez que NOT a une priorité plus grande que AND :
NOT Condition1 AND Condition2 sélectionnera les enregistrements ne satisfaisant pas la condition1 et satisfaisant la condition2, alors que NOT (Condition1 AND Condition2) sélectionnera les enregistrements ne satisfaisant pas les deux conditions 1 et 2. Une fois de plus, n'hésitez pas à mettre des parenthèses !

3.4.5 La clause LIKE

La clause LIKE permet de faire des recherches approximatives sur le contenu d'un champ. Par exemple, pour sélectionner les clients dont le nom commence par la lettre D :

```
SELECT *
FROM Clients
WHERE nom LIKE 'S*';
```

Tout comme dans les requêtes Access, le symbole * remplace un ensemble de caractères, pour représenter tous les noms commençant par S, on utilisera 'S*', tous ceux se terminant par S, on utilisera '*S', et tous ceux comportant la lettre S : '*S*'. Le symbole ? ne remplace qu'un seul caractère. Si on a deux clients nommés Dupond et Dupont, on utilisera 'Dupon?'.

*Attention : certaines versions de SQL n'utilisent pas les caractères * et ? mais d'autres caractères spécifiques, certaines versions utilisent notamment le caractère % à la place de *. Consultez donc la documentation de votre SGBD.*

3.4.6 Les jointures

La jointure va nous permettre de sélectionner des informations dans plusieurs tables grâce aux relations existant entre ces tables. Il va néanmoins falloir indiquer comment se fait la relation entre ces tables.

Par exemple : récupérer le nom et le prénom du client ayant passé la commande n°1 :

```
SELECT nom, prénom
FROM Clients, Commande
WHERE Commande.num_client = Client.num_client AND num_commande = 1;
```

La clause WHERE indique que le numéro de commande doit être égal à 1 et que la jointure va se faire sur le numéro de client : une fois que SQL va trouver la commande n° 1 dans la table commande, il va prendre le numéro de dient contenu dans l'enregistrement et avec ce numéro, aller chercher dans la table Clients le nom et le prénom correspondants à ce numéro.

Notez que lorsqu'on utilise plusieurs tables, il faut faire attention que deux tables n'aient pas de champs ayant le même nom, si c'est le cas, et pour les différencier, on utilise, comme dans l'exemple, la notation: table.nom_du_champ. Si on est sur que le nom ne se retrouvera pas dans plusieurs tables, on peut l'utiliser sans le préfixer avec le nom de la table.

3.4.7 Supprimer les doubles avec DISTINCT

Supposons que nous voulions la liste des clients ayant acheté quelque chose. Nous voulons que chaque client ayant acheté quelque chose ne soit affiché qu'une seule fois – nous ne voulons pas savoir ce qu'a acheté chaque client – nous voulons juste connaître les clients qui ont acheté quelque chose. Pour cela, nous allons devoir dire à SQL de supprimer les doubles du résultat de la sélection pour n'afficher les clients qu'une seule fois. Pour cela, nous allons utiliser la clause DISTINCT.

Nous allons d'abord faire une jointure entre les tables Clients et Commande, et ajouter la clause DISTINCT sur le champ ou la répétition peut se produire :

```
SELECT Client.num_client,nom, prénom
FROM Clients, Commande
WHERE Commande.num_client = Client.num_client;
```

Si on exécute cette requête directement, SQL va nous renvoyer une liste des numéros, prénom et nom correspondants aux noms et prénoms des clients ayant passé chaque commande, il est clair qu'un client ayant passé plusieurs commandes va se retrouver plusieurs fois dans cette liste.

```
SELECT DISTINCT Clients.num_client,nom, prénom
FROM Clients, Commande
WHERE Commande.num_client = Client.num_client AND num_commande = 1;
```

En indiquant la clause DISTINCT avant le champ num_client, on indique à SQL qu'on ne veut pas voir apparaître plusieurs fois un client ayant ce numéro dans la sélection renvoyée.

On peut même rendre le résultat de la sélection plus agréable à la lecture en utilisant la clause ORDERBY :

```
SELECT nom, prénom
FROM Clients, Commande
WHERE Commande.num_client = Client.num_client AND num_commande = 1
ORDER BY nom,prénom;
```

La sélection renvoyée va être classée alphabétiquement d'abord sur le nom, puis sur le prénom.

3.4.8 Les fonctions d'ensemble

SQL a cinq fonctions importantes : SUM, AVG, MAX, MIN et COUNT. On les appelle fonctions d'ensemble parce qu'elles résument le résultat d'une requête plutôt que de renvoyer une liste d'enregistrements.

Fonction	Signification
SUM ()	Donne le total d'un champ de tous les enregistrements satisfaisant la condition de la requête. Le champ doit bien sur être de type numérique
AVG ()	donne la moyenne d'un champ de tous les enregistrements satisfaisant la condition de la requête
MAX ()	donne la valeur la plus élevée d'un champ de tous les enregistrements satisfaisant la condition de la requête
MIN ()	Donne la valeur la plus petite d'un champ de tous les enregistrements satisfaisant la condition de la requête.
COUNT (*)	Renvoie le nombre d'enregistrements satisfaisant la requête.

Exemples :

```
SELECT
MIN(prix_unitaire),MAX(prix_unitaire),AVG(prix_unitaire)
```

Va retourner le prix le plus petit de la table Produit, le prix le plus élevé et le prix moyen.

```
SELECT COUNT (*)
FROM Produits
WHERE libelle LIKE 'P*';
```

Va retourner le nombre de produits dont le libellé commence par la lettre 'P'.

3.4.9 La clause GROUP BY

Une des utilisations les plus courantes de la clause GROUP BY est son association avec une fonction d'ensemble (le plus souvent COUNT, pour compter le nombre d'enregistrements dans chaque groupe). Par exemple, si nous voulons la liste des vendeurs, avec pour chaque vendeur le nombre de ventes qu'il a fait :

```
SELECT num_vendeur,COUNT (*)
FROM Commandes
GROUP BY num_vendeur;
```

On peut ajouter une condition à la requête, par exemple, la liste des vendeurs avec leur nombre de vente pour le mois de janvier.

```
SELECT num_vendeur,COUNT (*)
FROM Commandes
GROUP BY num_vendeur
HAVING mois(date)=1;
```

On utilisera pour cela la clause HAVING.

3.4.10 Les sous-requêtes

On peut imbriquer autant de requêtes que l'on veut. La condition après la clause WHERE peut porter sur le résultat d'une autre requête (ou sous-requête).

Supposons les tables suivantes :

Vente
Num_acheteur
Num_produit
Prix

Acheteurs
Num_acheteur
Nom
Prénom

Cette table contient, pour chaque acheteur, le produit qu'il a acheté et le prix d'achat.

Nous voulons la liste des acheteurs ayant acheté des articles chers. Nous considérerons qu'un article cher est un article dont le prix est supérieur à la moyenne du prix des produits achetés + 100 francs.

```
SELECT Num_acheteur
FROM Vente
WHERE prix >
      ( SELECT AVG (prix) + 100
      FROM Vente);
```

Vous pouvez constater que condition de la requête est basée sur le résultat d'une autre requête. Dans cet exemple, à chaque fois qu'un acheteur aura acheté un article cher, son numéro apparaîtra, pour éviter cela, on utilise la clause DISTINCT num_acheteur pour éliminer les doubles.

Autre exemple avec ces deux tables : nous savons qu'il y a une erreur sur l'orthographe du nom de l'acheteur du produit n° 1, il devrait s'appeler 'Dupont' :

```
UPDATE Acheteurs
SET nom = 'Dupont'
WHERE num_acheteur =
      (SELECT num_acheteur
      FROM Ventes
      WHERE num_article = 1);
```

N'oubliez pas cette règle à propos des sous-requêtes : Lorsque vous faites une sous-requête dans la clause WHERE, la clause SELECT de cette sous-requête doit avoir un nombre et des types de champs correspondants à ceux se trouvant après la clause WHERE de la requête principale. Autrement dit, si vous avez "WHERE champ = (SELECT ...);", le résultat du SELECT doit être un seul champ puisqu'il n'y a qu'un seul champ après le WHERE, ET leur type doit correspondre (les deux doivent être numériques ou être des chaînes de caractères, etc.), sinon, la requête ne renverra rien ou sortira avec une erreur selon les systèmes.