

## CRÉATION DE REQUÊTE EN SQL DANS ACCESS

Le SQL — Structured Query Language, Langage de Requêtes Structuré — est un langage permettant d'extraire des informations d'une base de données en fonction de critères. Grâce au SQL, il est ainsi possible de filtrer, modifier, ajouter, supprimer des données dans une base de données relationnelles. Des clauses (SELECT, FROM, WHERE, etc.) permettent de définir les critères pour choisir les données à conserver.

## ACCÈS AU SQL DANS ACCESS

Dans Access, lors de la création d'une requête, celle-ci est également disponible en langage SQL

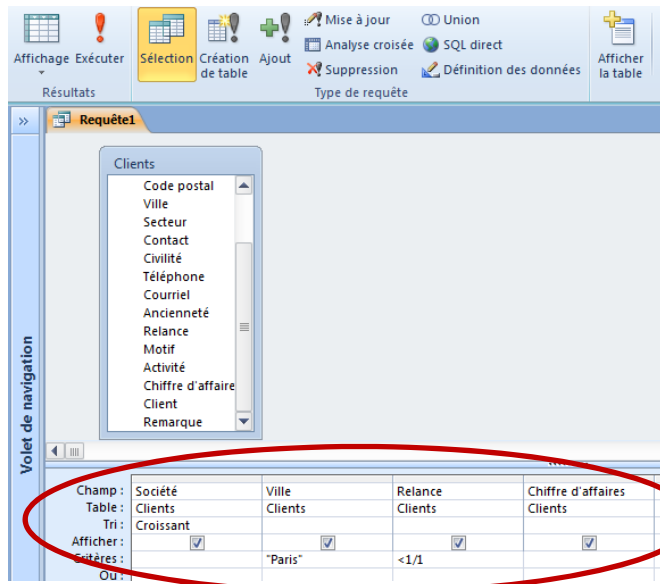
### CRÉATION D'UNE REQUÊTE AVEC L'OUTILS DE REQUÊTES D'ACCESS

#### ACCÈS À LA CRÉATION DE REQUÊTE

1. Onglet Créer  
Création de requêtes
2. Cliquer deux fois sur la table à ajouter (ou une fois et cliquer sur le bouton Ajouter)

#### CRÉER SA REQUÊTE

1. Faire glisser les champs à conserver sur les colonnes voulues
2. Dans la ligne « Tri », choisir Croissant ou Décroissant pour les champs voulus
3. Dans les lignes « Critères », saisir ses critères :



#### PENSER À SAUVEGARDER :

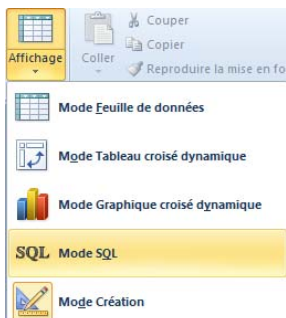
1. **Ctrl | S** ou cliquer sur le bouton Enregistrer (📁)
2. Saisir le nom de sa requête au clavier (exemple : *Requête SQL*) et **OK**

## ACCÉDER AU MODE SQL DE SA REQUÊTE

Onglet Accueil

Affichages

Mode SQL



- ✓ La requête apparaît en mode SQL avec différentes clauses comme **SELECT...** pour les champs à utiliser, **FROM...** pour le nom de la table à ouvrir, **WHERE** pour les critères créés, **ORDER BY...** pour les tris des champs, etc. :

```
Requête SQL
SELECT Clients.Société, Clients.Ville, Clients.Relance, Clients.[Chiffre d'affaires]
FROM Clients
WHERE (((Clients.Ville)="Paris") AND ((Clients.Relance)<#1/1/2017#))
ORDER BY Clients.Société;
```

Cette instruction codée en SQL peut être modifiée, allégée et complétée suivant ses besoins

- ✓ Dans Access, le code SQL se termine par un point-virgule (;), non obligatoire.

## POUR EXÉCUTER UNE REQUÊTE EN SQL

1. Onglet Créer  
Exécuter



2. Éventuellement, confirmer les demandes de validation si la requête modifie le contenu des tables
- ✓ Le résultat de la requête apparaît aussitôt.

## POUR QUITTER LE SQL ET REVENIR À L’AFFICHAGE DE SA REQUÊTE :

Onglet Accueil

Affichages

Mode Création : pour modifier sa requête

ou Mode Affichage des données : pour visualiser le résultat

## STRUCTURE ET ORDRE DES CLAUSES SQL

L’ordre des clauses SQL expliquées dans les chapitres qui suivent est très important. Voici un exemple de structure de tables liées avec des critères et un ordre de tri :

**SELECT DISTINCT** Liste des champs à afficher (Distinct permet d’éviter les doublons)

**FROM** Table principale **LEFT JOIN** Table secondaire **ON** Champ lié entre les deux tables

**WHERE** Comparaisons

**ORDER BY** Champs de tri

- ✓ Pour aérer son instruction SQL et la rendre plus lisible, il est possible d’aller à la ligne pour chaque clause.

## LES TYPES DE REQUÊTES DANS ACCESS

Les quelques instructions en SQL qui suivent peuvent être préparées en mode création de requêtes d'Access puis complétées et affinées en passant par le mode SQL (Onglet Accueil, Affichage, Mode SQL).

### AFFICHAGE DES CHAMPS VOULUS D'UNE TABLE

#### CODE SQL DE CHAMPS EN PROVENANCE D'UNE SEULE TABLE

Exemple de code qui affiche les champs *Société*, *CP* et *Date d'ancienneté* de la table *Clients* :

```
SELECT Clients.Société, Clients.CP, Clients.[Date d'ancienneté] FROM Clients;
```

1. **SELECT** : liste des champs à utiliser (champs *Société*, *Clients* et *Date d'ancienneté* de la table *Clients*), ceux-ci sont séparés par des virgules (,)
  - ✓ Si un champ contient un espace dans son nom, il devra être écrit entre crochet.  
Exemple : `[Date d'ancienneté]`
2. **FROM** : table ou requête à utiliser (ici, la table *Clients*)
  - ✓ S'il n'y a qu'une seule table, il n'est pas nécessaire de remettre le nom de la table devant chaque nom de champ : `SELECT Société, CP, [Date d'ancienneté] FROM Clients;`

#### ÉLIMINATION DES DOUBLONS AVEC DISTINCT

```
SELECT DISTINCT Intervention.Type FROM Intervention;
```

- **DISTINCT** : élimination des doublons sur les champs choisis
- **ALL** : récupère toutes les données (valeur par défaut, non nécessaire)

### FILTRE DES DONNÉES AVEC WHERE (OÙ)

Il est possible de ne récupérer que les données répondant à certains critères et de les trier sur l'un des champs.

#### STRUCTURE DU CODE SQL POUR FILTRER DES DONNÉES

```
SELECT Liste des champs à récupérer FROM Table à utiliser WHERE conditions;
```

#### Exemple

```
SELECT Type, [Montant facturé], [Date d'intervention] FROM Intervention WHERE Type="Formation";
```

**WHERE Type="Formation"** : ne garde les enregistrements que pour lesquels le champ *Type* contient le mot « Formation »

- ✓ Access a tendance à ajouter des parenthèses qui peuvent parfois être supprimées, exemple :  
...**WHERE** (((Intervention.[Type])="Formation")); → ...**WHERE** Intervention.[Type]="Formation";

#### ATTRIBUTS DE LA FONCTION WHERE

##### Opérateurs mathématiques de la clause WHERE

= : égal

> : supérieur

< : inférieur

>= : supérieur ou égal

<= : inférieur ou égal

<> : différent

### Exemple

```
SELECT Type, [Montant facturé] FROM Intervention WHERE [Montant facturé]>=5000;
```

Affiche les champs *Type* et *Montant facturé* de la table *Intervention*, uniquement pour les enregistrements dont le champ *Montant facturé* est supérieur ou égal à 5000.

### QUELQUES CLAUSES AVEC WHERE

#### Rechercher une chaîne de caractère dans un champ avec l'opérateur LIKE (*comme*)

```
SELECT Société, Ville FROM Clients WHERE [Ville] Like "**Paris*";
```

Affiche tous les enregistrements dont le champ *Ville* contient Paris (Paris, Paris Cédex, Paris-la-Défense mais aussi Corneilles-en-Parisis)

Attributs de l'opérateur *Like* :

- \* : remplace toute chaîne de caractère :  
*Like* "P\*" : noms commençant par P
- ? : remplace un seul caractère :  
*Like* "Dupon?" : trouve Dupont, Dupond, etc. mais pas les Duponton  
*Like* "43???" : Codes postaux dans le département 43  
(si le champ contenant le code postal est de type texte)
- # : remplace un chiffre
- *Like* "15/??/?????" : date tombant le 15 du mois
- [A-G] : ensemble de caractères :  
*Like* "[p-t]\*" : données commençant par P, Q, R, S, T, exemple : Paris, Strasbourg, Toulouse...
- [1-8] : plage de nombres :  
*Like* "[2-7]"; nombres 2, 3, 4, 5, 6 et 7
- [!A-G] ou [!1-8] : le pont d'exclamation exclue les caractères indiqués entre crochets :  
*Like* "[!p-t]\*" : exclu les données commençant par P, Q, R, S, T

#### Rechercher des valeurs comprises entre deux valeurs avec Between (*entre*)

```
SELECT [Chiffre d'affaires] FROM Clients WHERE [Chiffre d'affaires] Between 10000 And 50000;
```

Affiche les chiffres d'affaires compris entre 10 000 et 50 000 inclus

#### Recherche des noms dans une liste avec In (*dans*)

```
SELECT Société, Ville FROM Clients WHERE Ville In ("Paris","Toulouse","Orléans");
```

Affiche uniquement les enregistrements ayant les villes Paris, Toulouse et Orléans

#### Exclure des résultats avec Not (*ne pas*)

```
SELECT Société, Ville FROM Clients WHERE Not Ville="Paris";
```

Exclut les enregistrements dont la ville est Paris

✓ L'attribut *Not* peut se combiner avec *Between*, *In*, etc.

### COMBINAISON DE PLUSIEURS CONDITIONS

Il est possible de combiner plusieurs conditions avec les opérateurs AND, OR, XOR, etc.

### Exemple

```
SELECT Société, Relance FROM Clients WHERE Relance>=Date() And Relance<=Date()+15;
```

Date de relance comprise entre aujourd'hui et dans 15 jours après aujourd'hui (Date() donne la date du jour).

## Opérateurs logiques

- **AND** (et) : toutes les conditions doivent être vraies (cf. exemple précédent)
- **OR** (ou) : au moins une des conditions doit être vraie :  
... **WHERE** Entreprise **Like** "E\*" **OR** Entreprise **Like** "M\*" : entreprises commençant par un E ou par un M
- **XOR** (ou exclusif) : une et une seule des conditions doit être vraie :  
...**WHERE** Ville="Paris" **XOR** Activité="Technique" : recherche soit les enregistrements dont la ville est *Paris* soit ceux dont l'activité est *Technique* mais n'affichera pas les enregistrements dont la ville est *Paris* et l'activité est *Technique*

## TRIER DES DONNÉES AVEC ORDER BY (TRIÉ PAR)

Il est possible de trier ses données par ordre croissant ou décroissant sur le champ de son choix

### Exemple

```
SELECT Société, Ville FROM Clients ORDER BY Société
```

**ORDER BY** Société : trie les enregistrements de la base *Clients* par ordre alphabétique du champ *Société*

### Attributs de la clause ORDER BY

- **ASC** : tri croissant (valeur par défaut)  
...**ORDER BY** Ventes **ASC** : montant des ventes du plus petit au plus grand
  - **DESC** : tri décroissant  
...**ORDER BY** Relance **DESC** : date de la plus récente à la plus ancienne
- ✓ Il est possible de combiner plusieurs **ORDER BY** :  
...**ORDER BY** Nom, Prénom : tri par nom puis, en cas de deux noms identiques, par prénom

## SÉLECTION D'ENREGISTREMENT EN HAUT OU EN BAS D'UNE LISTE

Le SQL permet également de n'afficher que les premiers ou derniers enregistrements d'une liste

### Exemple de sélection des premiers enregistrements avec TOP (haut)

```
SELECT TOP 5 Société, [Chiffre d'affaires] FROM Clients;
```

Affiche les cinq premiers enregistrements de la table *Clients*

☞ Ces données restent triées dans l'ordre par défaut des enregistrements ; mais elles peuvent être triées dans l'ordre de son choix grâce à la clause **ORDER BY** (cf. section précédente), exemple :

```
SELECT TOP 3 Société, [Chiffre d'affaires] FROM Clients ORDER BY [Chiffre d'affaires];
```

Liste des sociétés ayant les trois chiffres d'affaires les plus élevés

☞ Il se peut que la commande SQL renvoie plus de trois enregistrements s'il y a des ex-aequo, exemple :

```
Société 1 : 1 000 000 €  
Société 2 : 800 000 €  
Société 3 : 500 000 €  
Société 4 : 500 000 €
```

### L'attribut PERCENT

Grâce à l'attribut **PERCENT**, il est possible d'avoir les premiers ou derniers enregistrements en pourcentage, exemple :

```
SELECT TOP 50 PERCENT Société, Ville FROM Clients ORDER BY Société ;
```

Affiche 50% des enregistrements de la table *Clients*

## SÉLECTION DE CHAMPS EN PROVENANCE DE PLUSIEURS TABLES AVEC LEFT JOIN (RELIER À GAUCHE)

La clause **FROM** peut être suivie d'attributs permettant de relier plusieurs tables entre elles :

```
SELECT Clients.Société, Clients.Ville, Formation.Cours, Formation.[Date début], Formation.Qté FROM  
Clients LEFT JOIN Formation ON Clients.N° = Formation.N_Client;
```

Il faut préciser les deux tables à utiliser et le champ commun entre les deux tables :

- **FROM** Clients : nom de la première base à utiliser (Clients)
  - **LEFT JOIN** Formation : sélectionne tous les enregistrements de la table mentionnée après **FROM** (Clients) avec uniquement ceux de la deuxième table à utiliser (Formation)  
**ON** Clients.N° = Formation.N\_Client : champs de la relation entre les deux tables ; le champ *N°* de la table *Clients* est relié (signe égal =) est lié au champ *N\_Client* de la table *Formation*.
- ✓ Il est possible d'ajouter plusieurs fois les clauses **LEFT JOIN** et **ON** suivant le nombre de tables à relier :
- ```
SELECT Clients.Société, Formation.Cours, Intervenants.Nom, Intervenants.Prénom FROM (Clients  
LEFT JOIN Formation ON Clients.N° = Formation.N_Client) LEFT JOIN Intervenants ON  
Formation.N_Formation = Intervenants.N_Formation;
```

La table *Clients* est reliée à la table *Formation* par les champs *N°* et *N\_Client* et la table *Formation* est reliée à la table *Intervenants* par le champ *N\_Formation* qui se trouve dans les deux tables

- ✓ Penser à mettre des parenthèses après **FROM** et ses champs liés pour définir le lien avec la première table

### Autres attributs de la clause FROM

L'attribut **LEFT JOIN** peut être remplacé par :

- **RIGHT JOIN** : affiche tous les enregistrements de la seconde table et seulement ceux qui sont liés dans la première table
- **INNER JOIN**: n'affiche que les enregistrements qui ont une correspondance dans l'autre table
- **IN** permet de récupérer des champs d'une table dans une base de données externe :  

```
SELECT Cours, Niveau FROM Formation IN "C:\Users\...\Suivi\OPCA.accdb";
```

Récupère les champs *Cours* et *Niveau* dans la table *Formation* de la base de données *OPCA.accdb*. Les guillemets et le chemin d'accès sont obligatoires.

### Élimination des doublons

Il se peut que les liens entre les tables entraînent l'affichage de doublons, pour les éviter, utiliser l'attribut **DISTINCTROW**. Exemple :

```
SELECT DISTINCTROW Clients.Société FROM Clients LEFT JOIN Formation ON Clients.N° =  
Formation.N_Client;
```

- ✓ **DISTINCTROW** est l'équivalent de **DISTINCT** pour les requêtes SQL sans liens entre les tables

## LES FONCTIONS DE REGROUPEMENT GROUP BY...HAVING (PAR GROUPE... AYANT)

Les fonctions de regroupement permettent de regrouper des données communes d'un champ afin d'effectuer un calcul en fonction de certains critères.

### EXEMPLE DE CALCUL AVEC UNE FONCTION LA REGROUPEMENT HAVING

Calcul de la somme du chiffre d'affaires par ville des sociétés de la table *Clients* ayant la communication comme activité :

```
SELECT Ville, Sum([Chiffre d'affaires]) AS [Total CA], Activité FROM Clients GROUP BY Ville, Activité HAVING Activité="Communication";
```

- **SELECT** Ville, Sum([Chiffre d'affaires]) : affichage du champ Ville et total (Sum) du champ *Chiffre d'affaire*
- **AS** [Total CA] : nom d'alias qui apparaîtra en haut de la colonne du calcul. Ce nom est choisi librement par l'utilisateur, sinon, par défaut, *Expr...* apparaîtra en haut de la colonne. **AS** peut être utilisé dans n'importe quelle clause SQL
- **FROM** Clients : champs en provenance de la table *Clients*
- **GROUP BY** Ville, Activité : champs qui devront être regroupés pour le calcul ou les tris
- **HAVING** Activité="Communication" : l'attribut **HAVING** permet de définir les critères des champs à conserver pour les regroupements. **HAVING** s'utilise comme **WHERE**.

### QUELQUES EXPRESSIONS DE REGROUPEMENT

- Sum() : Somme
- Avg() : Moyenne
- Min() : Minimum
- Max() : Maximum
- Count() : comptage du nombre d'enregistrements concernés, etc.

## METTRE À JOUR DES ENREGISTREMENTS AVEC LA CLAUSE UPDATE (METTRE À JOUR)

La clause **UPDATE** permet de modifier les données d'un ou de plusieurs champs. Le champ modifié et sa nouvelle valeur sont indiqués par l'attribut **SET**

### Modification d'un champ de texte

```
UPDATE Clients SET Motif = "Promotion" WHERE Ville="Paris";
```

Range « Promotion » dans le champ *Motif*, uniquement pour les enregistrements dont la ville est Paris pour la table *Clients*.

```
UPDATE Clients SET Prix = Prix*0.8 WHERE Activité="Communication";
```

Diminue le prix de 20% (100%-20%=80%) si l'activité est la communication

### EXEMPLE DE MISE À JOUR AVEC DES FILTRES DE DONNÉES SUR UNE SECONDE TABLE

Met une date dans 7 jours à partir d'aujourd'hui si le suivi est une livraison (dans la table *Cde*) et la ville est Paris (dans la table *Clients*)

```
UPDATE Clients LEFT JOIN Cde ON Clients.N°=Cde.N_Client SET Cde.[Date suivi] = Date()+7 WHERE Cde.Suivi="Livraison" AND Clients.Ville ="Paris"
```

## SUPPRIMER DES ENREGISTREMENTS AVEC LA CLAUSE DELETE (*SUPPRIMER*)

---

La clause **DELETE** permet de supprimer des enregistrements répondant à certains critères, exemple :  
**DELETE FROM** Clients **WHERE** Courriel **Is Null**;

Supprime tous les enregistrements qui n'ont pas de courriel (Is Null)

## INSÉRER DES NOUVEAUX ENREGISTREMENTS AVEC LA CLAUSE INSERT INTO (*INSÉRER DANS*)

---

La clause **INSERT INTO** permet de créer de nouveaux enregistrements en définissant les données à ajouter dans ces enregistrements

### Exemple :

La table *Clients* contient la liste des clients et la table *Formation*, la liste des formations. La fonction SQL suivante va donc permettre d'ajouter de nouvelles formations (dans la table *Formation*) avec un numéro de client, une date et une formation (« Formation Internet ») uniquement aux clients dont l'activité est la communication (d'après la table *Clients*) :

```
INSERT INTO Formation (N_Client, [Date début], Cours) SELECT N°, #3/20/2017#, "Formation Internet" FROM Clients WHERE Activité="Communication";
```

- **INSERT INTO** Formation : ajout d'enregistrements dans la table formation
- (**N\_Client**, [**Date début**], **Cours**) : champs qui vont recevoir des données dans les nouveaux enregistrements (les autres resteront vides)
- **SELECT** N°, #03/20/2017#, "Formation Internet" : données qui rempliront les champs précédemment définis. L'ordre de ces données correspond à l'ordre des champs précédents,
  - #03/20/2017# est une date au format *mois/jour/année* (ou écrire *Date()* pour la date du jour),
  - N° est un champ de la table *Clients* définie après le **FROM**. Ainsi, le champ *N\_Client* de la table *Formation* recevra la valeur correspondante du champ N° de la table *Clients*.
- **FROM** Clients : table source contenant les données qui seront envoyées dans la table de récupération (*Formation*)
- **WHERE** Activité="Communication" : seuls les enregistrements de la table *Clients* dont l'activité est la communication seront concernés par l'ajout d'enregistrements.

## CRÉATION D'UNE TABLE

---

### CRÉATION D'UNE NOUVELLE TABLE À PARTIR DE TABLES EXISTANTES AVEC SELECT... INTO (*SÉLECTIONNER DANS*)

Création d'une nouvelle table appelée *FormationParis* à partir des champs *N°*, *Société*, *Ville* de la table *Clients* et [*Date début*], *Cours* de la table *Formation*, si la ville est Paris

```
SELECT Clients.N°, Clients.Société, Clients.Ville, Formation.[Date début], Formation.Cours INTO FormationParis FROM Clients LEFT JOIN Formation ON Clients.N° = Formation.N_Client WHERE Clients.Ville="Paris";
```

- **SELECT...** : choix des champs de la nouvelle table
- **INTO** FormationParis : nom de la nouvelle table
- **FROM** Clients : table principale dont sont issues les données à ajouter à la nouvelle table
- **LEFT JOIN** Formation : table secondaire liée à la table principale
- **ON** Clients.N° = Formation.N\_Client : Champs de liaison entre les deux tables
- **WHERE...** : critère de récupération des données



## CRÉATION D'UNE TABLE VIDE AVEC CREATE TABLE (CRÉER UNE TABLE)

**CREATE TABLE** permet de créer la structure d'une table en définissant le nom de la table, les champs à créer, leur taille, le type de champ et les index souhaités.

Exemple, création de la table *Livraison* avec les champs *N\_Livraison*, *Contact*, *DateLivraison*, *Quantité*, *Montant*, *DossierClos* et *Remarque* avec le champ *N\_Gestion* comme clé primaire :

```
CREATE TABLE Livraison (N_Livraison COUNTER, Contact TEXT(40), DateLivraison DATE, Quantité  
NUMBER, Montant CURRENCY, DossierClos BIT, Remarque LONGTEXT,  
PRIMARY KEY(N_Livraison));
```

- **CREATE TABLE** Livraison : nom de la table à créer
- Mettre les champs entre parenthèse et définir leur type :
  - COUNTER : Numérotation automatique (NuméroAuto)
  - TEXT(40) : texte avec 40 caractères maximum
  - DATE : date
  - NUMBER : Nombre (ou INTEGER pour des petits nombres)
  - CURRENCY : monétaire
  - BIT : logique (oui/non)
  - LONGTEXT : champ mémo, etc.
- **PRIMARY KEY** (N\_Suivi) : contraint le champ N\_Suivi à être la clé primaire de la table.

## MODIFICATION DES CHAMPS D'UNE TABLE AVEC ALTER TABLE (MODIFIER LA TABLE)

La clause **ALTER TABLE** permet d'ajouter ou de supprimer des champs dans une table :

```
ALTER TABLE Suivi ADD Organisation TEXT(40);
```

- **ADD** : crée le champ *Organisation*, de type texte avec 40 caractères dans la table *Suivi*

```
ALTER TABLE Suivi DROP DossierClos;
```

- **DROP** : supprime le champ *DossierClos* de la table *Suivi*


## LANCEMENT AUTOMATIQUE D'UNE REQUÊTE SQL À PARTIR DU VBA

Dans Access, il est possible de créer des macros en VBA qui s'exécutent à partir d'un bouton placé dans un formulaire. Le code VBA de la macro peut contenir une instruction en SQL.

Voici un exemple de macro qui diminue de 20% les prix de vente des sociétés de la table *Clients* qui sont dans la communication

Code SQL : **UPDATE** Clients **SET** Prix = Prix\*0.8 **WHERE** Activité="Communication";



## CRÉATION DU BOUTON QUI VA LANCER LA REQUÊTE

1. Dans le formulaire voulu, passer en mode Création (Onglet Accueil, Affichage, Mode Création)
2. Onglet Création,  
dans *Contrôle*, cliquer sur le bouton *Bouton* (  )
3. Dans le formulaire, tracer un rectangle qui va contenir le bouton et cliquer sur **Annuler** pour fermer la fenêtre

4. Changer le titre du bouton qui apparaîtra dans le formulaire en cliquant deux fois dessus (exemple : Filtre SQL)
  5. Dans la fenêtre des propriétés du bouton (Onglet *Création, Feuille de propriétés* pour la faire apparaître), changer le nom du bouton :  
Onglet *Autres* de la feuille des propriétés,  
dans *Nom*, saisir un nom de bouton sans espace, exemple : BoutonSQL
- ✓ Ce nom est celui qui sera utilisé dans le code VBA alors que le titre précédent est celui qui apparaîtra dans le formulaire, visible de tous les utilisateurs.

## ASSOCIER UN CODE VBA AU BOUTON

---

1. Cliquer sur le bouton voulu
2. Onglet *Événements* de la feuille des propriétés,  
Sur clic  
Cliquer sur le bouton des générateurs (  )  
Choisir *Générateur de code* et 

## SAISIE DU CODE VBA CONTENANT LA CLAUSE SQL

---

Dans la fenêtre VBA qui apparaît avec le nom du bouton, saisir le code VBA avec la commande SQL voulue entre *Private Sub ...* et *End Sub* :

```
Private Sub BoutonSQL_Click()  
    DoCmd.RunSQL "UPDATE Clients SET Prix = Prix*0.8 WHERE Activité='Communication'"  
End Sub
```

## EXPLICATIONS DE LA COMMANDE DOCMD.RUNSQL "..."

- *DoCmd.RunSQL "..."* : commande qui exécute une instruction SQL
  - *UPDATE Clients* : Table *Clients*
  - *SET Prix = Prix\*0.8* : réduction du prix de 20% (100%-80%=20%)
  - *WHERE Activité='Communication'* : cette mise à jour ne concerne que les sociétés dont l'activité est la communication
- ✓ Comme l'attribut de la commande *DoCmd.RunSQL* nécessite de mettre le code est entre guillemets, les comparaisons ou les attributs qui sont déjà entre guillemets peuvent être mis entre des apostrophes :
- ```
DoCmd.RunSQL "UPDATE Clients SET Prix = Prix*0.8 WHERE Activité='Communication'"
```

## EXÉCUTION DE LA MACRO

---

1. Afficher le formulaire en mode formulaire (Onglet *Accueil, Affichage, Mode formulaire*)
2. Cliquer sur le bouton créé
3. La macro se lance et affiche un avertissement annonçant la mise à jour de plusieurs enregistrements

## LIENS CONSEILLÉS SUR LE LANGAGE SQL

Pour apprendre la structure des différentes commandes en SQL :

<https://access.developpez.com/sql/>

Présentation des attributs de la fonction Like :

[loufab.developpez.com/tutoriels/access/operateurlike](http://loufab.developpez.com/tutoriels/access/operateurlike)

La clause INSERT INTO :

<http://frederic.redonnet.free.fr/processus10/cours/courshtml/chap55reqactioncorr.htm>

Création et modification des tables :

<http://cerig.pagora.grenoble-inp.fr/tutoriel/bases-de-donnees/chap18.htm>

## EXERCICES SUR LES REQUÊTES EN SQL

### CRÉATION DES TABLES EN SQL

1. Créer la première table suivante appelée *Utilisateurs* avec les champs suivants (50 caractères pour *Société* et 5 pour le *Code postal*) et saisir les données :

N°	Société	Code postal	Ventes
1	Euréco	92088	40 000,00 €
2	A&C	75008	75 000,00 €
3	Nivat	75010	1 000,00 €
4	Mallay	59000	13 000,00 €
5	Larfeul	31000	12 000,00 €
6	Keller	31000	18 000,00 €
7	Gasquet	75002	5 000,00 €
8	Zain	31000	12 000,00 €
10	Turenne	75012	17 000,00 €
11	Comborn	95240	40 000,00 €

2. Créer une deuxième table qui sera appelée *Suivi*, avec les champs (25 caractères pour les produits) et les données suivantes :

N_Ventes	N_Client	Produit	Quantité
1	1	Ordinateur	25
2	2	Imprimante	10
3	10	Logiciel	18
4	2	Logiciel	30
5	4	Logiciel	12
6	5	Imprimante	6
7	9	Imprimante	22
8	11	Imprimante	18
9	7	Ordinateur	9
10	6	Logiciel	7

### CRÉATION DES REQUÊTES EN SQL

Créer les codes SQL qui suivent à partir des requêtes d'Access puis les exécuter

#### REQUÊTES SQL

Créer, avec une commande SQL, une requête appelée « Requête Ventes » qui affiche les champs *Société*, *Code postal*, *Ventes* de la table *Utilisateurs*, dont les montants des ventes sont compris entre 10 000 et 20 000 euros et afficher le résultat.

Créer, avec une commande SQL, une nouvelle requête appelée « Requête Logiciel » qui affiche, dans le champ *Société* de la table *Utilisateurs* et les champs *Produit* et *Quantité* de la table *Suivi* dont les produits sont des imprimantes

#### REQUÊTES MISES À JOUR

Créer une requête SQL de mise à jour, appelée « Requête remise » qui réduit de 10% les ventes de la table *Utilisateurs* pour les quantités supérieures ou égales à 15.

#### REQUÊTE SUPPRESSION

Créer une requête appelée « Requête suppression » qui supprime, dans la table *Utilisateurs* les sociétés ayant moins de 10 000 euros de ventes

#### REQUÊTE AJOUT D'ENREGISTREMENTS

Créer une requête appelée « Requête Ajout » qui ajoute « Clavier » dans le champ *Produit* avec une quantité de 1 dans la table *Suivi* pour les sociétés dont le code postal commence par 75.

## CORRECTION DES EXERCICES SUR LES REQUÊTES EN SQL

### ACCÈS ET EXÉCUTION DES REQUÊTES EN SQL

---

#### ACCÈS À LA REQUÊTE

##### Création de la requête

1. Onglet Créer  
Création de requêtes
2. Cliquer deux fois sur la table à ajouter (ou une fois et cliquer sur le bouton Ajouter)
3. Éventuellement, faire glisser les champs voulus dans la zone de critères, en bas
- ✓ Penser à sauvegarder et à donner un nom à sa requête

##### Accès au mode SQL

4. Onglet Accueil  
Affichages  
Mode SQL
5. Saisir la requête voulue en SQL

##### Exécution de la requête SQL

Onglet Créer  
Exécuter



### CRÉATION DES TABLES

---

#### REQUÊTE SQL POUR LA TABLE *UTILISATEURS*

**CREATE TABLE** Utilisateurs (N° COUNTER, Société TEXT(50), Ventes CURRENCY, PRIMARY KEY(N°))

- **CREATE TABLE** Utilisateurs Création de la table *Utilisateurs*
- avec les champs *N°* en numérotation automatique COUNTER, *Société* en texte avec 50 caractères TEXT(50), *Ventes* en monétaire CURRENCY
- **PRIMARY KEY(N°)** : N° est la clé primaire

#### REQUÊTE SQL DE CRÉATION DE LA TABLE *SUIVI*

**CREATE TABLE** Suivi (N\_Ventes COUNTER, N\_Client NUMBER, Produit TEXT(25), Quantité NUMBER, PRIMARY KEY(N\_Ventes))

- **CREATE TABLE** Suivi : création de la table *Suivi*
- avec les champs N\_Ventes (numérotation automatique), N\_Client (nombre), Produit (texte avec 25 caractères), Quantité (Nombre)
- **PRIMARY KEY(N\_Ventes)** : affectation de la clé primaire au champ *N\_Ventes*

## CRÉATION DES REQUÊTES

---

### REQUÊTES SQL

#### Requête de filtre « Requête Ventes » affichant les ventes compris entre 10 000 et 20 000 euros

```
SELECT Société, [Code postal], Ventes FROM Utilisateurs WHERE Ventes>=10000 AND Ventes<=20000;
```

- **SELECT** Société, [Code postal], Ventes : champ à afficher dans la requête
- **FROM** Utilisateurs : table d'origine des champs
- **WHERE** Ventes>=10000 **AND** Ventes<=20000 : Ventes comprises entre 10 000 et 20 000 euros

#### Requête utilisant les tables Utilisateurs et Suivi et affichant les données des deux tables dont le produit est « Imprimante »

```
SELECT Utilisateurs.Société, Suivi.Produit, Suivi.Quantité FROM (Utilisateurs LEFT JOIN Suivi ON Utilisateurs.N° =Suivi.N_Client) WHERE Suivi.Produit="Imprimante";
```

- **FROM** (Utilisateurs : table principale (*Utilisateurs*))
- **LEFT JOIN** Suivi : table secondaire (*Suivi*)
- **ON** Utilisateurs.N° =Suivi.N\_Client) : lien entre le champ N° de la table *Utilisateur* et le champ *N\_Client* de la table *Suivi*
- **WHERE** Suivi.Produit="Imprimante" : ne garde que les produits qui sont des imprimantes

#### REQUÊTE MISES À JOUR QUI RÉDUIT DE 10% LE MONTANT DES VENTES

```
UPDATE Utilisateurs INNER JOIN Suivi ON Utilisateurs.N° = Suivi.N_Client SET Utilisateurs.ventes = [Ventes]*0.9 WHERE Suivi.Quantité>=15;
```

- **UPDATE** Utilisateurs : mise à jour de la table *Utilisateurs*
- **SET** Utilisateurs.Ventes = Utilisateurs.Ventes\*0.9 : réduit de 10% la valeur du champ *Ventes*

#### REQUÊTE SUPPRESSION DES MONTANTS INFÉRIEURS À 10 000 EUROS

```
DELETE FROM Utilisateurs WHERE Ventes < 10000;
```

#### REQUÊTE AJOUT D'ENREGISTREMENTS

```
INSERT INTO Suivi ( N_Client, Produit, Quantité ) SELECT DISTINCTROW Utilisateurs.N°, "Clavier" , 1 FROM Utilisateurs LEFT JOIN Suivi ON Utilisateurs.N° = Suivi.N_Client WHERE Utilisateurs.[Code postal] Like "75*";
```

- **INSERT INTO** Suivi ( N\_Client, Produit, Quantité ) **SELECT DISTINCTROW** Utilisateurs.N°, "Clavier" , 1 : création de nouveaux enregistrements dans la table *Suivi* et ajoute à *N\_Client* la valeur du champ N° de la table *Utilisateur*, « Clavier » dans le champ *Produit* et 1 au champ *Quantité*
- **DISTINCTROW** : élimine les doublons de la table *Suivi* pour éviter d'ajouter autant de nouveaux champs qu'il existe d'enregistrements dans la table *Suivi*
- **FROM** Utilisateurs **LEFT JOIN** Suivi **ON** Utilisateurs.N° = Suivi.N\_Client : relie les tables *Utilisateurs* et *Suivi* avec le champ N° de la table *Utilisateurs* et le champ *N\_Client* de la table *Suivi*
- **WHERE** Utilisateur.[Code postal] **Like** "75???" : requête pour les codes postaux commençant par 75, **Like** "75\*" afficherait tous les départements commençant par 75, y compris 7500. ■