

# Le Langage SQL appliqué à Access

par Alexandre le Grand

Date de publication :

Dernière mise à jour :

Découvrez la syntaxe SQL d'Access. Au sommaire : définition et manipulation des données

1 - Les requêtes.....	3
1.1 - Commandes de base.....	3
1.1.1 - Les attributs placés après l'instruction SELECT.....	3
1.1.2 - Les attributs de la clause FROM.....	4
1.1.3 - Les paramètres de la clause WHERE.....	4
1.1.4 - Les mots réservés de la clause WHERE.....	5
1.1.5 - Opérateurs mathématiques de la clause WHERE.....	5
1.2 - Les commandes avancées.....	5
1.2.1 - Les fonctions de regroupement : GROUP BY...HAVING.....	5
1.2.2 - La clause ORDER BY.....	6
1.2.3 - La déclaration WITH OWNERACCESS OPTION.....	6
2 - Création de tables.....	7
2.1 - Instruction pour la création.....	7
2.1.1 - Création d'une table : CREATE TABLE.....	7
2.1.2 - La clause : CONSTRAINT.....	7
2.1.2.a - Index monochamp.....	7
2.1.2.b - Index Multichamp.....	7
2.2 - Instruction pour la modification.....	8
2.2.1 - Modification d'une table : ALTER TABLE.....	8
2.2.2 - Suppression d'une table.....	8
2.2.3 - Suppression d'enregistrements : DELETE.....	8
2.2.4 - Mise à jour d'enregistrement : UPDATE.....	8
2.2.5 - Insertion d'enregistrement : INSERT INTO.....	9
2.2.5.a - Insertion de valeurs.....	9
2.2.5.b - Insertion à partir d'une autre table.....	9
2.2.6 - Accès et sécurité.....	9

## 1 - Les requêtes

### 1.1 - Commandes de base

SELECT...FROM...WHERE

SELECT <Champ>, <Champ2>,..... FROM <Relation> WHERE <Condition>

\* **Remarque1** : <Relation> peut être une ou plusieurs tables ou requêtes.

\* **Remarque2** : Si <Champ> est issu d'une seule table le nom du Champ suffit si ce n'est pas le cas l'écriture est la suivante Table.Champ ou Table.

\* **Remarque3** : pour les Champs et les tables avec espaces <N° de client> utiliser les crochets [N ° de client].

\* **Remarque4** : Si la sélection porte sur tous les champs on peut utiliser SELECT \* FROM...WHERE

#### 1.1.1 - Les attributs placés après l'instruction SELECT

SELECT (ALL, DISTINCT, DISTINCTROW, TOP, PERCENT)

Ces attributs se plaçant avant l'énumération des champs, portent sur les enregistrements (tuples) eux mêmes. Ils permettent d'opérer une sélection sur les enregistrements qui vont faire l'objet de la clause WHERE.

**ALL** : sélectionnent tous les enregistrements (commande par défaut si omission).

**DISTINCT** : sélectionne les enregistrements pour lesquels il n'y a pas de doublon dans la projection réalisée (champs choisis).

Exemple :

Voici une table où il existe des doublons pour certains champs, mais où aucun enregistrement entier (N°, Nom, Prénom, Age) ne se répète.

Projection de Nom et Age : SELECT DISTINCT Personne.Nom, Personne.Age FROM Personne;

Explication : Un des trois DUPONT n'apparaît pas du fait de l'attribut DISTINCT. En effet pour les champs choisis il existe un doublon. Pour voir les trois DUPONT apparaître il aurait fallu ajouter le champ Prénom dans la requête. DISTINCT opère donc uniquement sur les champs sélectionnés.

**DISTINCTROW** : cet attribut concerne les requêtes qui utilisent une jointure entre deux tables. La sélection des enregistrements d'une table mère jointe à une table fille duplique les enregistrements autant de fois qu'ils leur correspondent un enregistrement dans la table fille. Pour éviter cet inconvénient on aurait pu utiliser DISTINCT mais si la sélection fait apparaître des doublons dans sa projection (doublons dans les champs choisis) certains enregistrements qui pourtant sont uniques de par leur identifiant vont être omis.

**DISTINCTROW** permet de palier à cet inconvénient car il porte sur les enregistrements et non les champs. Autrement dit, DISTINCTROW omet les doublons qui, pour une table, apparaissent du fait de sa jointure avec une autre.

**DISTINCTROW** ne fonctionne que si la requête porte sur au moins deux tables et qu'une seule est concernée dans la projection.

**Exemple** :

Table <personne> jointe avec table <enfants>

SELECT DISTINCTROW Personne.Nom, Personne.Age FROM Personne INNER JOIN Enfants ON Personne.[N°] = enfants.[N°Prop];

Les doublons DUPONT Charles pour la table Personne ont été omis même si la projection fait apparaître de nouveaux doublons dans les champs choisis (<DUPONT>, <35>). En effet au regard des enregistrements de la table ils ne sont pas en double l'un possède le N° 1 et son prénom est Charles, l'autre le N° 2 et son prénom est Fred.

L'attribut DISTINCT omettrait ces doublons issus de la projection.

**TOP** : Cet attribut sélectionne un certain nombre d'enregistrements qui se trouvent en haut ou en bas d'une plage déterminée par une clause ORDER BY.

**PERCENT** : PERCENT sélectionne un certain pourcentage d'enregistrements qui figurent en haut ou en bas d'une plage définie par une clause ORDER BY

Remarque sur les instructions SELECT et FROM: on peut utiliser un pseudonyme pour les champs et les tables en employant AS (AS peut être omis) :

Exemple : SELECT DISTINCT Personne.Nom, Personne.Age AS [Nombre d'années] FROM Personne AS Pers;

## 1.1.2 - Les attributs de la clause FROM

- **IN** : permet de sélectionner des champs dans une base externe.

Exemple : SELECT DISTINCT Personne.Nom, Personne.Age FROM personne IN [C:\...\Base.mdb];

- **INNER JOIN** <Table> ON <Champ1>=<Champ2> : permet de sélectionner les enregistrements de deux tables jointes en n'affichant pour les deux tables que les enregistrements qui ont une correspondance pour leur champ commun (clé primaire et clé étrangère).

- **LEFT JOIN** <Table> ON <Champ1>=<Champ2> : permet de sélectionner les enregistrements de deux tables jointes en affichant pour la table de gauche (clé primaire) tous les enregistrements même s'ils n'ont pas de correspondance dans la table de droite (clé étrangère).

- **RIGHT JOIN** <Table> ON <Champ1>=<Champ2> : permet de sélectionner les enregistrements de deux tables jointes en affichant pour la table de droite (clé étrangère) tous les enregistrements même s'ils n'ont pas de correspondance dans la table de gauche (clé primaire).

```
SELECT Notes.Note FROM Notes WHERE (Notes.Note)>11 And (Notes.Note)<13;
```

```
SELECT Notes.Note FROM Notes WHERE Notes.Note=11 Or Notes.Note=13;
```

```
SELECT Notes.Note FROM Notes WHERE Notes.Note>11 Xor Notes.Note<13;
```

```
SELECT Notes.Note FROM Notes WHERE Notes.Note>10 And Not notes.Note=12;
```

## 1.1.3 - Les paramètres de la clause WHERE

Cette clause permet une restriction suivant une ou plusieurs critères.

Les critères peuvent être accumulés par les opérateurs logiques AND, OR, XOR, Eqv, Not, Imp

**AND** : <Expr1> AND <Expr2> ==> les deux expressions doivent être vérifiées.

**OR** : <Expr1> OR <Expr2> ==> ou inclusif, au moins une des deux expressions doit être vérifiée (et/ou).

**XOR** : <Expr1> XOR <Expr2> ==> ou exclusif, au plus une des deux expressions doit être vérifiée (ou seulement).

**Eqv** : <Expr1> Eqv <Expr2> ==> est utilisé pour effectuer une équivalence logique sur deux expressions.

**Not** : <Expr1> -opérateur- NOT <Expr2> ==> effectue la négation d'un opérateur ( AND Not, OR Not...).

**Imp** : <Expr1> Imp <Expr2> ==> est utilisé pour effectuer une implication logique sur deux expressions.

Exemples :

```
SELECT Notes.Note FROM Notes WHERE (Notes.Note)>11 And (Notes.Note)<13;
```

## 1.1.4 - Les mots réservés de la clause WHERE

**BETWEEN** : sélectionne plusieurs valeurs comprises entre deux valeurs. Si ces deux valeurs sont de type texte **BETWEEN** sélectionne par rapport à l'ordre alphabétique.

Exemple :

```
SELECT Notes.Note FROM Notes WHERE Notes.Note BETWEEN 12 And 15;
```

**IN** : sélectionne des valeurs qui appartiennent à une liste

Exemple :

```
SELECT Elève.Nom, Elève.Prénom FROM Elève WHERE Elève.Nom In ("Marc", "Maxime");
```

**IN** peut se combiner avec **NOT** pour exclure des valeurs

**LIKE** permet de chercher une occurrence dans les champs d'une table.

- \* = n'importe quelle chaîne de caractères
- ? = n'importe quel caractère # = n'importe quel nombre
- [A-G] = plage de caractères
- [1-8] = plage de nombres
- [!A-G] = hors plage de caractères
- [!1-8] = hors plage de nombres
- En SQL non Access % = \* et \_ = ?

## 1.1.5 - Opérateurs mathématiques de la clause WHERE

=, >, <, >=, <=, <>.

## 1.2 - Les commandes avancées

SELECT...FROM...WHERE...GROUP BY...HAVING...ORDER BY...WITH OWNERACCESS OPTION

### 1.2.1 - Les fonctions de regroupement : GROUP BY...HAVING

Cette Clause est incontournable en SQL puisqu'elle permet d'effectuer un calcul sur un champ donné. Imaginons par exemple que l'on voudrait savoir la moyenne de chaque élève. Ce calcul nécessite un regroupement des notes par élève. Une requête classique projeterait chaque note de chaque élève, des doublons apparaîtraient pour le champ Elève.Nom. Il faut donc indiquer au moteur de base de données qu'un regroupement est nécessaire, pour qu'il puisse dans un premier temps effectuer la somme des notes pour un élève données, puis dans un second temps diviser le résultat par le nombre de notes.

Ce regroupement est possible par la clause **GROUP BY**. Une fois le regroupement réalisé, il ne reste plus qu'indiquer qu'elle opération l'on veut réaliser après la clause **SELECT**.

La clause **HAVING** est similaire à **WHERE** à ceci près qu'elle supporte des expressions de regroupement (moyenne, compte...) alors que **WHERE** ne les supporte pas. En effet **HAVING** peut sélectionner les enregistrements dont la projection réalisée par **SELECT** dépend d'un regroupement **GROUP BY**.

Exemples :

- On voudrait connaître tous les élèves qui ont trois notes

```
SELECT DISTINCTROW Elève.Nom, Elève.Prénom, Count(Notes.Note) AS [Nombre de Notes]
FROM Elève INNER JOIN Notes ON Elève.[N°] = Notes.[N°]
GROUP BY Elève.Nom, Elève.Prénom HAVING Count(Notes.Note)=3;
```

On voudrait connaître le nombre d'élèves qui ont une note maximum de 14

```
SELECT DISTINCTROW Count(Elève.Nom) AS [Nombre d'élèves], Max(Notes.Note) AS [Note maximum]
FROM Elève INNER JOIN Notes ON Elève.[N°] = Notes.[N°]
GROUP BY Notes.note HAVING Max(Notes.Note)=14;
```

Remarque1 : La clause GROUP BY doit être précédée d'au moins un champ projeté par SELECT (si le champ en question possède une expression de regroupement, cette expression n'est pas mentionnée).

Remarque2 : la clause HAVING doit porter sur un ou plusieurs champs projetés par SELECT, les expressions de regroupement sont concernées.

Les expressions de regroupement :

- \* **Moyenne** (Avg)
- \* **Compte** (Count)
- \* **Premier et dernier** (First,Last)
- \* **Minimum et maximum** (Min, Max)
- \* **Ecart-type** (StDev), **EcartypeP** (StDevP) > StDev : écart-type d'échantillon. **StDevP** : écart-type de population.
- \* **Somme** (Sum)
- \* **Variante** (Var,VarP) > Var : variance d'échantillon. VarP : Variance de population.

## 1.2.2 - La clause ORDER BY

ORDER BY champ1 ASC ou DESC , champ2 ASC ou DESC ...

ASC = Croissant ; DESC = Décroissant > par défaut ASC.

Elle sert à classer un ou plusieurs domaines de valeur pour un champ donnée. Si une table est liée à un autre par une relation 1,N on peut effectuer plusieurs classement. Par exemple on veut la liste des noms des élèves dans l'ordre alphabétique et leurs notes dans l'ordre décroissant.

Exemple :

```
SELECT DISTINCTROW Elève.Nom, Notes.Note
FROM Elève INNER JOIN Notes ON Elève.[N°] = Notes.[N°]
ORDER BY Elève.Nom, Notes.Note DESC;
```

## 1.2.3 - La déclaration WITH OWNERACCESS OPTION

Dans un environnement Multi-Utilisateurs où figure un groupe de travail protégé, la déclaration WITH OWNERACCESS OPTION permet dans le cadre d'une requête d'accorder à l'utilisateur qui exécute cette requête les mêmes autorisations qu'au propriétaire de la requête.

## 2 - Création de tables

Le langage SQL permet de créer des tables (fichiers permettant l'enregistrement de données classées par champ ayant chacun un type (numérique, texte...) et une taille. Certains moteurs en capsulent les tables dans un fichier unique (Access) d'autres créent autant de tables que de fichiers. SQL permet aussi de modifier des tables créées.

### 2.1 - Instruction pour la création

#### 2.1.1 - Création d'une table : CREATE TABLE

**CREATE TABLE** NomTable (<Champ1 Type (taille)> **CONSTRAINT** <Index>, <Champ2 Type (taille)> **CONSTRAINT** <Index2>)...**CONSTRAINT** <Index\_multichamp>

\* **NomTable** : Nom de la table à créer.

\* **Champ1, Champ2** : Nom du ou des champs à créer dans la nouvelle table (au moins un champ est nécessaire)..

\* **Type** : Type de données de champ dans la nouvelle table > Texte (Text), Mémo (Memo), Numérique (Byte, Integer, Long, Single...) Date/heure (Date ou Time), Monétaire (Currency), NuméroAuto, Oui/Non (YesNo), Objet OLE (OleObject).

\* **Taille** : La taille du champ en caractères (champs Texte et Binaires uniquement).

\* **Index1, Index2** : Paramètres de la clause **CONSTRAINT** définissant un index monochamp

\* **Index\_multichamp** : Paramètres de la clause **CONSTRAINT** définissant un index multichamp

Remarque sur les index : un index permet de d'identifier un champ pour faciliter les recherches du moteur de base de données. Le fait qu'il soit unique (sans doublon) ou pas (avec doublons) permet de créer des types de relations différentes avec un ou plusieurs champs d'une autre table.

\* Table1 (Champ1 IndexUNIQUE) <> Table2 (Champ2 IndexUNIQUE) 1,1

\* Table1 (Champ1 IndexUNIQUE) <> Table2 (Champ2 IndexPasUNIQUE) 1,N

\* Table1 (Champ1 IndexUNIQUE) <> Table2 (Champ2 IndexPasUNIQUE) N,1

\* Table1 (Champ1 IndexPasUNIQUE) <> Table2 (Champ2 IndexPasUNIQUE) M,N

Une clé primaire est un index particulier puisqu'elle identifie de manière unique non pas un champ mais un enregistrement, de la même manière qu'un index elle peut définir plusieurs relations.

#### 2.1.2 - La clause : CONSTRAINT

##### 2.1.2.a - Index monochamp

**CONSTRAINT** <Nom> **PRIMARY KEY** et/ou **UNIQUE** et/ou **NOT NULL REFERENCES** <Table\_externe> (<Champ\_externe1>, <Champ\_externe2>.....)

\* **Nom** : Nom de la contrainte.

\* **PRIMARY KEY, UNIQUE, NOT NULL** caractéristique de l'index.

\* **Table\_externe** : table contenant la clé étrangère dans le cas d'une relation.

\* **Champ\_externe1, Champ\_externe2** : nom du ou des champs ayant pour clé externe le champ indexé.

##### 2.1.2.b - Index Multichamp

**CONSTRAINT** <Nom> **PRIMARY KEY** (<Primaire1>, <Primaire2> ...) **UNIQUE** (<Unique1>, <Unique2> ...) **NOT NULL** (<NotNull1>, <NotNull2>...) **FOREIGN KEY** (<Réf1>, <Réf2> ...) **REFERENCES** <Table\_externe> (<Champ\_externe1>, <Champ\_externe2>...)

\* **Nom** : Nom de la contrainte

\* **PRIMARY KEY, UNIQUE, NOT NULL** : caractéristique de l'index.

\* **Unique1, Unique2** : nom du ou des champs excluant les doublons.

\* **NotNull1, NotNull2** : nom du ou des champs interdisant des valeurs nulles.

\* **Ref1, Ref2** : nom du ou des champs étant la clé externe d'une relation.

\* **Table\_externe** : table contenant la clé étrangère dans le cas d'une relation.

\* **Champ\_externe1, Champ\_externe2** : nom du ou et champs de la table externe ayant pour clé externe Ref1 et Ref2.

## 2.2 - Instruction pour la modification

### 2.2.1 - Modification d'une table : ALTER TABLE

```
ALTER TABLE <NomTable> [ADD [COLUMN <champ type (taille)> CONSTRAINT <Index>  
CONSTRAINT Index_multichamp] DROP [COLUMN <champ> CONSTRAINT <Nom_d'index>]]
```

### 2.2.2 - Suppression d'une table

**DROP TABLE** <NomTable> ou **INDEX** <index> **ON** <NomTable>

Exemples :

```
DROP TABLE MaTable; ou DROP INDEX MonIndex ON MaTable
```

### 2.2.3 - Suppression d'enregistrements : DELETE

L'instruction **DELETE** permet de créer une requête Suppression qui supprime les enregistrements répondant à la clause WHERE d'une ou plusieurs tables parmi celles qui figurent dans la clause FROM.

**DELETE** <NomTableSource.\*> **FROM** <NomTable> **WHERE** <critère>

NomTableSource.\* désigne la table à partir de laquelle s'effectue la suppression dans le cas d'une jointure avec une autre table. En effet si la suppression porte sur la table source les enregistrements dans la table but seront supprimer. Si le cas contraire, la suppression porte sur la table but seuls les enregistrements de cette table seront supprimer. Si la suppression porte sur une table l'astérisque suffit.

Exemples : Table Employé jointe 1,1 avec Table Salaire

```
DELETE salaire.* FROM Employés INNER JOIN Salaire ON Employés.NumEmployé = Salaire.NumEmployé WHERE  
Employés.Nom="morin";
```

### 2.2.4 - Mise à jour d'enregistrement : UPDATE

L'instruction **UPDATE** permet de créer une requête Mise à jour qui modifie les valeurs des champs d'une table spécifiée en fonction des critères déterminés.

```
UPDATE <NomTable> SET <Nouvelle_valeur> WHERE critère
```

Remarque pour l'instruction **UPDATE**, la clause WHERE peut s'employer avec toutes les commandes que l'on peut trouver dans les requêtes sélection.

Table Employé Jointe 1,1 avec Table Salaire.

Table DIRECTION : Directrice, Président, Sous-Directrice

Exemples :

Un nouveau Directeur prend les commandes de l'entreprise et décide d'augmenter de 20% les salaires inférieurs à 6000 pour les employés qui ont 30 ans ou plus.

```
UPDATE Employés INNER JOIN Salaire ON Employés.NumEmployé = Salaire.NumEmployé
SET Salaire = Salaire * (1 + (20/100))
WHERE Salaire < 6000 AND Age >= 30 WITH OWNERACCESS OPTION;
```

```
UPDATE DIRECTION SET Nom = "PEUPLUS", Prénom = "Jean", Age = 45 WHERE DIRECTION.Nom="ZEUBLOUZ" WITH
OWNERACCESS OPTION;
```

## 2.2.5 - Insertion d'enregistrement : INSERT INTO

### 2.2.5.a - Insertion de valeurs

```
INSERT INTO <NomTable> (<Champ>, <Champ2>#..) VALUES (<Valeur1>, <Valeur2>#..)
```

### 2.2.5.b - Insertion à partir d'une autre table

```
INSERT INTO <NomTableDestination> SELECT <Champ1>, <Champ2>#. FROM <NomTableSource> WHERE <condition>
```

Pour que l'insertion soit valide il faut que les champs de la table source aient le même type et la même longueur que la table destination.

## 2.2.6 - Accès et sécurité

Cette instruction n'est pas valide sous Access, mais fait partie de SQL Oracle.

```
GRANT [[SELECT, UPDATE, INSERT, DELETE, ALTER, INDEX] ALL] ON <NomTable> TO NomUtilisateur
```

