Syntaxe SQL (les bases)

Select
SELECT "nom de colonne" FROM "nom de table"

Distinct
SELECT DISTINCT "nom de colonne"
FROM "nom de table"

Where
SELECT "nom de colonne"
FROM "nom de table"
WHERE "condition"

And/Or
SELECT "nom de colonne"
FROM "nom de table"
WHERE "condition simples"
{[AND|OR] "condition simples"}

Order By
SELECT "nom de colonne"
FROM "nom de table"
[WHERE "condition"]
ORDER BY "nom de colonne" [ASC, DESC]

Count
SELECT COUNT("nom de colonne")
FROM "nom de table"

Insert Into
INSERT INTO "nom de table" ("colonne 1", "colonne 2", ...)
valeurS ("valeur 1", "valeur 2", ...)

Update
UPDATE "nom de table"
SET "colonne 1" = [nouvelle valeur]
WHERE {condition}

Delete From
DELETE FROM "nom de table"
WHERE {condition}

1. SELECT

1.1. SQL SELECT

À quoi servent les commandes SQL ? Une utilisation courante de ces commandes consiste à sélectionner les données de tables contenues dans une base de données. Nous venons tout juste de mentionner deux mots-clés : il faut **SELECT** (SÉLECTIONNER) des informations **FROM** (PROVENANT) d'une table. (Notez qu'une table est un conteneur qui réside dans la base de données où sont stockées les données). Nous avons ainsi la structure SQL la plus basique :

SELECT "nom de colonne" FROM "nom de table"

Pour illustrer l'exemple ci-dessus, supposons que nous avons la table suivante :

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	1500 €	05-Jan-1999
San Diego	250 €	07-Jan-1999
Los Angeles	300 €	08-Jan-1999
Boston	700 €	08-Jan-1999

Cette table sera utilisée comme exemple tout au long du didacticiel (elle apparaîtra dans toutes les sections). Pour sélectionner tous les magasins dans cette table, il faut saisir :

SELECT store_name **FROM** Store_Information

Résultat :

store_name
Los Angeles
San Diego
Los Angeles
Boston

Plusieurs noms de colonne et de table peuvent être sélectionnés.

1.2. SQL WHERE

Il est ainsi possible de sélectionner conditionnellement les données d'une table. Par exemple, en utilisant le mot-clé **WHERE** il est possible de retrouver uniquement des magasins dont les ventes sont supérieures à 1 000 €. La syntaxe est comme suit :

SELECT "nom de colonne"
FROM "nom de table"
WHERE "condition"

Par exemple, pour sélectionner tous les magasins dont les ventes sont supérieures à 1 000 € dans la Table ***Store_Information***,

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	1500 €	05-Jan-1999
San Diego	250 €	07-Jan-1999
Los Angeles	300 €	08-Jan-1999
Boston	700 €	08-Jan-1999

il faut saisir :

```
SELECT store_name
FROM Store_Information
WHERE Sales > 1000
```

Résultat :

store_name
Los Angeles

1.3. SQL AND OR

Dans la section précédente, nous avons vu que le mot-clé **WHERE** peut s'utiliser pour sélectionner conditionnellement des données d'une table. Cette condition peut être une condition simple (comme celle de la section précédente), ou combinée. Les conditions combinées se constituent de plusieurs conditions simples connectées par **AND** ou **OR**. Une seule instruction SQL peut contenir un nombre illimité de conditions simples.

La syntaxe d'une condition combinée est comme suit :

```
SELECT "nom de colonne"
FROM "nom de table"
WHERE "condition simples"
{[AND|OR] "condition simples"}+
```

1.4. SQL ORDER BY

Jusqu'à présent, nous avons vu comment extraire des données d'une table à l'aide des commandes **SELECT** et **WHERE**. Il convient souvent de lister les résultats dans un ordre particulier. Le classement peut se faire dans un ordre ascendant ou descendant, ou peut être établi par des valeurs saisies en chiffres ou en lettres. Dans de tels cas, il est possible d'utiliser le mot-clé **ORDER BY** pour atteindre notre objectif.

La syntaxe d'une instruction **ORDER BY** est comme suit :

```
SELECT "nom de colonne"
FROM "nom de table"
[WHERE "condition"]
ORDER BY "nom de colonne" [ASC, DESC]
```

2. INSERT

2.1. SQL INSERT INTO

Dans les sections précédentes, nous avons appris comment retrouver des informations dans les tables. Mais comment pouvons-nous ajouter ces lignes de données dans ces tables en première position ? Ce thème et l'instruction **INSERT** seront décrits dans cette section, tandis que l'instruction **UPDATE** sera décrite dans la section suivante.

Sous SQL, il existe deux manières de base pour **INSÉRER** des données dans une table : l'une consiste à insérer des données une ligne à la fois, et l'autre plusieurs à la fois. Pour **INSÉRER** des données une ligne à la fois :

La syntaxe pour l'insertion de données dans une table une ligne à la fois est comme suit :

```
INSERT INTO "nom de table" ("colonne 1", "colonne 2", ...)
VALUES ("valeur 1", "valeur 2", ...)
```

Supposons que nous avons une table qui a la structure suivante,

Table ***Store_Information***

Column Name	Data Type
store_name	char(50)
Sales	float
Date	datetime

et pour insérer une ligne supplémentaire dans la table représentant les données de ventes pour Los Angeles le 10 janvier 1999, dont les ventes de ce magasin, à ce jour, s'élevaient à 900 €, il faudra utiliser le script SQL suivant :

```
INSERT INTO Store_Information (store_name, Sales, Date)
VALUES ('Los Angeles', 900, '10-Jan-1999')
```

3. UPDATE

3.1. SQL UPDATE

Nous pouvons, parfois, être amenés à modifier les données contenues dans une table. Pour ce faire, il convient d'utiliser la commande **UPDATE**. La syntaxe de cette commande est :

```
UPDATE "nom de table"
SET "colonne 1" = [nouvelle valeur]
WHERE {condition}
```

Dans le cas d'une table comme suit :

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	1500 €	05-Jan-1999
San Diego	250 €	07-Jan-1999
Los Angeles	300 €	08-Jan-1999
Boston	700 €	08-Jan-1999

et nous nous rendons compte que les ventes pour Los Angeles du 08-Jan-1999 sont en réalité de 500 € au lieu de 300 €, et que cette entrée particulière doit être corrigée. Pour ce faire, nous utiliserons la requête SQL suivante :

```
UPDATE Store_Information
SET Sales = 500
WHERE store_name = "Los Angeles"
AND Date = "08-Jan-1999"
```

La table résultante ressemblerait à

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	1500 €	05-Jan-1999
San Diego	250 €	07-Jan-1999
Los Angeles	500 €	08-Jan-1999
Boston	700 €	08-Jan-1999

Dans ce cas, il n'y a qu'une ligne qui satisfait la condition de la clause **WHERE**. Toutes les lignes qui satisfont la condition seront modifiées.

Il est également possible de **METTRE À JOUR** plusieurs colonnes à la fois. La syntaxe dans ce cas-ci ressemblerait à ce qui suit :

```
UPDATE "nom de table"
SET colonne 1 = [valeur 1], colonne 2 = [valeur 2]
WHERE {condition}
```

4. DELETE

4.1. SQL DELETE FROM

Nous pouvons, parfois, être amenés à supprimer des enregistrements d'une table. Pour ce faire, il est possible d'utiliser la commande **DELETE FROM**. La syntaxe de cette commande est :

```
DELETE FROM "nom de table"
WHERE {condition}
```

Il est plus facile de comprendre en utilisant un exemple. Supposons que nous avons actuellement une table comme suit :

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	1500 €	05-Jan-1999
San Diego	250 €	07-Jan-1999
Los Angeles	300 €	08-Jan-1999
Boston	700 €	08-Jan-1999

et nous décidons de ne conserver aucune information de Los Angeles dans cette table. Pour ce faire, nous saisissons la requête SQL suivante :

```
DELETE FROM Store_Information
WHERE store_name = "Los Angeles"
```

Le contenu de la table devrait paraître à

Table ***Store_Information***

store_name	Sales	Date
San Diego	250 €	07-Jan-1999
Boston	700 €	08-Jan-1999