Mémento SQL – MySQL – Web Dynamique

SELECT

L'utilisation la plus courante de SQL consiste à lire des données issues de la base de données. Cela s'effectue grâce à la commande SELECT, qui retourne des enregistrements dans un tableau de résultat. Cette commande peut sélectionner une ou plusieurs colonnes d'une table.

Exemple

Imaginons une base de données appelée « client » qui contient des informations sur les clients d'une entreprise.

Table « client » :

identifiant	prenom	nom	ville
1	Pierre	Dupond	Paris
2	Sabrina	Durand	Nantes
3	Julien	Martin	Lyon
4	David	Bernard	Marseille
5	Marie	Leroy	Grenoble

Si l'on veut avoir la liste de toutes les villes des clients, il suffit d'effectuer la requête SQL ci-dessous :

```
SELECT ville FROM client
```

De cette manière on obtient le résultat suivant :

ville
Paris
Nantes
Lyon

ville
Marseille
Grenoble

Obtenir plusieurs colonnes

Avec la même table client il est possible de lire plusieurs colonnes à la fois. Il suffit tout simplement de séparer les noms des champs souhaités par une virgule. Pour obtenir les **prénoms** et les **noms** des **clients** il faut alors faire la requête suivante:

SELECT prenom, nom FROM client

Ce qui retourne ce résultat:

prenom	nom
Pierre	Dupond
Sabrina	Durand
Julien	Martin
David	Bernard
Marie	Leroy

WHERE

La commande WHERE dans une requête SQL permet d'extraire les lignes d'une base de données qui respectent une condition. Cela permet d'obtenir uniquement les informations désirées.

Syntaxe

La commande WHERE s'utilise en complément à une requête SELECT. La façon la plus simple de l'utiliser est la suivante:

```
SELECT nom_colonnes FROM nom_table WHERE condition
```

Exemple

Imaginons une base de données appelée « client » qui contient le nom des clients, le nombre de commandes qu'ils ont effectués et leur ville:

id	nom	nbr_commande	ville
1	Paul	3	paris
2	Maurice	0	rennes
3	Joséphine	1	toulouse
4	Gérard	7	paris

Pour obtenir seulement la liste des clients qui habitent à Paris, il faut effectuer la requête suivante:

```
SELECT * FROM client WHERE ville = 'paris'
```

Cette requête retourne le résultat suivant:

id	nom	nbr_commande	ville
1	Paul	3	paris
4	Gérard	7	paris

Attention: dans notre cas tout est en minuscule donc il n'y a pas eu de problème. Cependant, une table est sensible à la casse, il faut faire attention aux majuscules et minuscules.

Opérateurs de comparaisons

Il existe plusieurs opérateurs de comparaisons. La liste ci-jointe présente quelques-uns des opérateurs les plus couramment utilisés.

Opérateur	Description
=	Égale
<>	Pas égale
!=	Pas égale

Opérateur	Description
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égale à
<=	Inférieur ou égale à
IN	Liste de plusieurs valeurs possibles
BETWEEN	Valeur comprise dans un intervalle donnée
LIKE	Recherche en spécifiant le début, milieu ou fin d'un mot.
IS NULL	Valeur est nulle
IS NOT NULL	Valeur n'est pas nulle

ORDER BY

La commande ORDER BY permet de trier les lignes dans un résultat d'une requête SQL. Il est possible de trier les données sur une ou plusieurs colonnes, par ordre ascendant ou descendant.

Syntaxe

Une requête où l'on souhaite filtrer l'ordre des résultats utilise la commande ORDER BY de la sorte :

```
SELECT colonne1, colonne2  FROM table  ORDER BY colonne1
```

Par défaut les résultats sont classés par ordre ascendant, toutefois il est possible d'inverser l'ordre en utilisant le suffixe DESC après le nom de la colonne. Par ailleurs, il est possible de trier sur plusieurs colonnes en les séparant par une virgule. Une requête plus élaborée ressemblerait à cela :

```
SELECT colonne1, colonne2, colonne3  FROM table  ORDER BY colonne1 DESC, colonne2 ASC
```

A noter : ce n'est pas obligatoire d'utiliser le suffixe « ASC » sachant que les résultats sont toujours classés par ordre ascendant par défaut. Toutefois, c'est plus pratique pour mieux s'y retrouver, surtout si on a oublié l'ordre par défaut.

Exemple

Pour l'ensemble de nos exemples, nous allons prendre une base « utilisateur » de test, qui contient les données suivantes :

id	nom	prenom	date_inscription	tarif_total
1	Durand	Maurice	2012-02-05	145
2	Dupond	Fabrice	2012-02-07	65
3	Durand	Fabienne	2012-02-13	90
4	Dubois	Chloé	2012-02-16	98
5	Dubois	Simon	2012-02-23	27

Pour récupérer la liste de ces utilisateurs par ordre alphabétique du nom de famille, il est possible d'utiliser la requête suivante :

```
SELECT *  FROM utilisateur  ORDER BY nom
```

Résultat :

id	nom	prenom	date_inscription	tarif_total
4	Dubois	Chloé	2012-02-16	98
5	Dubois	Simon	2012-02-23	27
2	Dupond	Fabrice	2012-02-07	65
1	Durand	Maurice	2012-02-05	145
3	Durand	Fabienne	2012-02-13	90

INSERT INTO

L'insertion de données dans une table s'effectue à l'aide de la commande INSERT INTO. Cette commande permet au choix d'inclure une seule ligne à la base existante ou plusieurs lignes d'un coup.

Insertion d'une ligne à la fois

Pour insérer des données dans une base, il y a 2 syntaxes principales :

- Insérer une ligne en indiquant les informations pour chaque colonne existante (en respectant l'ordre)
- Insérer une ligne en spécifiant les colonnes que vous souhaitez compléter. Il est possible d'insérer une ligne renseignant seulement une partie des colonnes.

Insérer une ligne en spécifiant toutes les colonnes

La syntaxe pour remplir une ligne avec cette méthode est la suivante :

```
INSERT INTO table VALUES ('valeur 1', 'valeur 2', ...)
```

Cette syntaxe possède les avantages et inconvénients suivants :

- Obliger de remplir toutes les données, tout en respectant l'ordre des colonnes
- Il n'y a pas le nom de colonne, donc les fautes de frappe sont limitées. Par ailleurs, les colonnes peuvent être renommées sans avoir à changer la requête
- L'ordre des colonnes doit rester identique sinon certaines valeurs prennent le risque d'être complétée dans la mauvaise colonne

Insérer une ligne en spécifiant seulement les colonnes souhaitées

Cette deuxième solution est très similaire, excepté qu'il faut indiquer le nom des colonnes avant « VALUES ». La syntaxe est la suivante :

```
INSERT INTO table (nom_colonne_1, nom_colonne_2, ...)   VALUES ('valeur 1', 'valeur 2', ...)
```

A noter : il est possible de ne pas renseigner toutes les colonnes. De plus, l'ordre des colonnes n'est pas important.

Insertion de plusieurs lignes à la fois

Il est possible d'ajouter plusieurs lignes à un tableau avec une seule requête. Pour ce faire, il convient d'utiliser la syntaxe suivante :

```
INSERT INTO client (prenom, nom, ville, age)
VALUES   ('Rébecca', 'Armand', 'Saint-Didier-des-Bois', 24),
('Aimée', 'Hebert', 'Marigny-le-Châtel', 36),
('Marielle', 'Ribeiro', 'Maillères', 27),
('Hilaire', 'Savary', 'Conie-Molitard', 58);
```

A noter : lorsque le champ à remplir est de type VARCHAR ou TEXT il faut indiquer le texte entre guillemet simple. En revanche, lorsque la colonne est un numérique tel que INT ou BIGINT il n'y a pas besoin d'utiliser de guillemet, il suffit juste d'indiquer le nombre.

Un tel exemple sur une table vide va créer le tableau suivant :

id	prenom	nom	ville	age
1	Rébecca	Armand	Saint-Didier-des-Bois	24
2	Aimée	Hebert	Marigny-le-Châtel	36
3	Marielle	Ribeiro	Maillères	27
4	Hilaire	Savary	Conie-Molitard	58

SQL UPDATE

La commande UPDATE permet d'effectuer des modifications sur des lignes existantes. Très souvent cette commande est utilisée avec WHERE pour spécifier sur quelles lignes doivent porter la ou les modifications.

Syntaxe

La syntaxe basique d'une requête utilisant UPDATE est la suivante :
UPDATE table SET nom_colonne_1 = 'nouvelle valeur' WHERE *condition*

Cette syntaxe permet d'attribuer une nouvelle valeur à la colonne nom_colonne_1 pour les lignes qui respectent la condition stipulée avec WHERE. Il est aussi possible d'attribuer la même valeur à la colonne nom_colonne_1 pour toutes les lignes d'une table si la condition WHERE n'était pas utilisée.

A noter, pour spécifier en une seule fois plusieurs modification, il faut séparer les attributions de valeur par des virgules. Ainsi la syntaxe deviendrait la suivante :

UPDATE table SET colonne_1 = 'valeur 1', colonne_2 = 'valeur 2', colonne_3 = 'valeur 3' WHERE *condition*

Exemple

Imaginons une table « client » qui présente les coordonnées de clients.

Table « client » :

id	nom	rue	ville	code_postal	pays
1	Chantal	12 Avenue du Petit Trianon	Puteaux	92800	France
2	Pierre	18 Rue de l'Allier	Ponthion	51300	France
3	Romain	3 Chemin du Chiron	Trévérien	35190	France

Modifier une ligne

Pour modifier l'adresse du client Pierre, il est possible d'utiliser la requête SQL suivante :

```
UPDATE client  SET rue = '49 Rue Ameline',
ville = 'Saint-Eustache-la-Forêt', code_postal = '76210'
WHERE id = 2
```

Cette requête sert à définir la colonne rue à « 49 Rue Ameline », la ville à « Saint-Eustache-la-Forêt » et le code postal à « 76210 » uniquement pour ligne où l'identifiant est égal à 2.

Résultats :

id	nom	rue	ville	code_postal	pays
1	Chantal	12 Avenue du Petit Trianon	Puteaux	92800	France
2	Pierre	49 Rue Ameline	Saint-Eustache-la-Forêt	76210	France
3	Romain	3 Chemin du Chiron	Trévérien	35190	France

Modifier toutes les lignes

Il est possible d'effectuer une modification sur toutes les lignes en omettant d'utiliser une clause conditionnelle. Il est par exemple possible de mettre la valeur « FRANCE » dans la colonne « pays » pour toutes les lignes de la table, grâce à la requête SQL ci-dessous.

```
UPDATE client  SET pays = 'FRANCE'
```

Résultats :

id	nom	rue	ville	code_postal	pays
1	Chantal	12 Avenue du Petit Trianon	Puteaux	92800	FRANCE
2	Pierre	49 Rue Ameline	Saint-Eustache-la-Forêt	76210	FRANCE
3	Romain	3 Chemin du Chiron	Trévérien	35190	FRANCE

DELETE

La commande DELETE en SQL permet de supprimer des lignes dans une table. En utilisant cette commande associée à WHERE il est possible de sélectionner les lignes concernées qui seront supprimées.

Attention

Avant d'essayer de supprimer des lignes, il est recommandé d'effectuer une **sauvegarde de la base de données**, ou tout du moins de la table concernée par la suppression. Ainsi, s'il y a une mauvaise manipulation il est toujours possible de restaurer les données.

Syntaxe

La syntaxe pour supprimer des lignes est la suivante :
DELETE FROM `table` WHERE condition

Attention : s'il n'y a pas de condition WHERE alors **toutes** les lignes seront supprimées et la table sera alors vide.

Exemple

Imaginons une table « utilisateur » qui contient des informations sur les utilisateurs d'une application.

Table « utilisateur » :

id	nom	prenom	date_inscription
1	Bazin	Daniel	2012-02-13
2	Favre	Constantin	2012-04-03
3	Clerc	Guillaume	2012-04-12

id	nom	prenom	date_inscription
4	Ricard	Rosemonde	2012-06-24
5	Martin	Natalie	2012-07-02

Supprimer une ligne

Il est possible de supprimer une ligne en effectuant la requête SQL suivante :

```
DELETE FROM `utilisateur`  WHERE `id` = 1
```

Une fois cette requête effectuée, la table contiendra les données suivantes :

id	nom	prenom	date_inscription
2	Favre	Constantin	2012-04-03
3	Clerc	Guillaume	2012-04-12
4	Ricard	Rosemonde	2012-06-24
5	Martin	Natalie	2012-07-02

Supprimer plusieurs lignes

Si l'on souhaite supprimer les utilisateurs qui se sont inscrit avant le **10/04/2012**, il va falloir effectuer la requête suivante :

```
DELETE FROM `utilisateur`  WHERE `date_inscription` < '2012-04-10'
```

La requête permettra alors de supprimer les utilisateurs « Daniel » et « Constantin ». La table contiendra alors les données suivantes :

id	nom	prenom	date_inscription
3	Clerc	Guillaume	2012-04-12
4	Ricard	Rosemonde	2012-06-24

id	nom	prenom	date_inscription
5	Martin	Natalie	2012-07-02

Il ne faut pas oublier qu’il est possible d’utiliser d’autres conditions pour sélectionner les lignes à supprimer.