MySQL: la commande SELECT

Savoir interroger sa base

L'écriture de requêtes est tout un art dès lors que la complexité d'une base de données devient plus ou moins grande. Les mauvais temps de réponses de beaucoup de sites proviennent souvent de requêtes mal formées et/ou non optimales. Bien souvent une simple réécriture de celles−ci améliore considérablement la réactivité d'un site sans avoir à mettre en oeuvre toute une série de technologies palliatives tels les caches par exemple. Nous allons donc regarder ici de plus près cette fameuse commande SELECT qui en effraye plus d'un.

Syntaxe de SELECT

Le cas SELECT *

En théorie déconseillé car SELECT * demande de charger tous les champs d'un enregistrement, son utilisation est parfaitement justifiable dans le cas ou la taille totale de ce qui sera sélectionné est acceptable. Une clause WHERE limitant le nombre d'enregistrements candidats pourra ainsi justifier de l'usage de SELECT *.

SELECT DISTINCT

DISTINCT permet d'éliminer les doublons dans une réponse, en effet par défaut, un SELECT seul est identique à un SELECT ALL. Soit la table *table1*:

table1	
NOM	**PRENOM**
Dupont	Jean
Durand	Martin
Laurier	Jean

Le query suivant:

```
SELECT PRENOM
FROM table1;
```

aura comme résultat:

NOM
Jean
Martin
Jean

et le query suivant:

```
SELECT DISTINCT PRENOM
FROM table1;
```

aura comme résultat:

NOM
Jean
Martin

CONCAT()

La concaténation permet d'agréger plusieurs champs en un seul. Normallement, en SQL cet opérateur est représenté ainsi **||** mais très peu de SGBD l'implémentent. MySQL nous fournit la fonction CONCAT(). L'utilisation de concat est très intéressante dans le cas où l'on doit afficher des informations synthétiques plutôt que de réaliser l'opération sois−même à l'intérieur d'un script PHP: de manière générale, tout ce qui pourra être réalisé à l'intérieur d'une requête sera toujours préférable à une solution faisant appel à un traitement ultérieur par un langage de script y compris PHP.

En reprenant la table *table1*, le query suivant:

```
SELECT CONCAT(NOM,'−',PRENOM)
FROM table1;
```

aura comme résultat:

CONCAT(NOM,'−',PRENOM)
Dupont−Jean
Durand−Martin
Laurier−Jean

CONCAT() AS

L'utilisation de la fonction CONCAT() donne pour résultat un champ dont le nom est celui de l'opération réalisée (voir ci−dessus). Pour des raisons pratiques évidentes il est préférable de pouvoir nommer explicitement le champ résultant. On utilise pour cela l'opérateur **AS** qui permet de définir un *alias* pour le champ. En reprenant l'exemple précédent, nous pourrions écrire le query ainsi:

```
SELECT CONCAT(NOM,'−',PRENOM) AS Identity
FROM table1;
```

qui aurait comme résultat:

Identity
Dupont−Jean
Durand−Martin
Laurier−Jean

MySQL: la commande SELECT

La clause WHERE

La clause **WHERE** permet de réaliser un test logique sur la condition qui la suit. Un résultat est obtenu lorsque la condition évaluée est VRAIE. Une condition utilise les opérateurs de comparaison et les opérateurs logiques dans le but de restreindre une recherche.

Soit la table *table2*:

table2		
ID	**NOM**	**PRENOM**
1	Dupont	Jean
2	Durand	Martin
4	Laurier	Jean
7	Gombart	Henry
8	Androux	Michel

Le query suivant:

```
SELECT *
FROM table2
WHERE id=7;
```

aura pour résultat:

ID	NOM	PRENOM
7	Gombart	Henry

alors que le query suivant:

```
SELECT *
FROM table2
WHERE PRENOM='Jean';
```

aura comme résultat:

ID	NOM	PRENOM
1	Dupont	Jean
4	Laurier	Jean

et le query suivant:

```
SELECT *
FROM table1
WHERE ID > 0;
```

aura comme résultat la table en son entier.

La clause ORDER BY

Jusqu'à présent les résultats retournés n'étaient pas forcément classés. La clause **ORDER BY** permet de spécifier un ordre croissant ou décroissant par rapport à un ou plusieurs champs. Par défaut une clause **ORDER BY** *champ1* est idantique à **ORDER BY** *champ1* **ASC**. Si l'on prend comme exemple la table *table3* suivante:

table3		
ID	**NOM**	**PRENOM**
1	Dupont	Jean
2	Dupont	Alain
4	Laurier	Jean
7	Gombart	Henry
8	Androux	Michel

le query suivant:

```
SELECT * FROM table3
ORDER BY NOM;
```

est identique au query:

```
SELECT * FROM table3
ORDER BY NOM ASC;
```

et aura comme résultat:

ID	NOM	PRENOM
8	Androux	Michel
1	Dupont	Jean
2	Dupont	Alain
7	Gombart	Henry
4	Laurier	Jean

et le query:

```
SELECT * FROM table3
ORDER BY NOM DESC,PRENOM ASC;
```

aura comme résultat:

ID	NOM	PRENOM
4	Laurier	Jean
7	Gombart	Henry
2	Dupont	Alain
1	Dupont	Jean
8	Androux	Michel

La clause GROUP BY

La clause **GROUP BY** permet de regrouper des valeurs d'un champ en vue d'un traitement statistique. Cette clause n'est donc pratiquement jamais utilisée seule mais en conjonctions avec des fonctions que nous allons étudier ci−dessous.

COUNT()

La fonction **COUNT()** permet comme son nom l'indique de compter le nombre de lignes. Utilisée en association avec la clause **GROUP BY** elle permet de compter les lignes satisfaisant un critère et regroupées selon les valeurs d'un champ.

Soit la table *table4* stockant les scores d'un jeu en ligne pour chaque utilisateur:

table4		
ID	**LOGIN**	**SCORE**
1	alain	12
2	alain	14
3	alain	9
4	pitou	17
5	pitou	14

Nombre de parties par joueur:

```
SELECT LOGIN,COUNT(*) AS nbGAMES
FROM table4
GROUP BY LOGIN;
```

aura pour résultat:

LOGIN	nbGAMES
alain	3
pitou	2

AVG()

la fonction **AVG()** permet de calculer une moyenne. Ainsi, le query suivant:

```
SELECT LOGIN,AVG(SCORE) AS MOY
FROM table4 GROUP BY LOGIN
ORDER BY MOY DESC;
```

aura comme résultat:

LOGIN	MOY
pitou	15.5000
alain	11.6667

MIN()

La fonction **MIN()** renvoie la valeur minimum d'une série. Appliqué à la table *table4* le query suivant:

```
SELECT LOGIN,MIN(SCORE) AS MINI
FROM table4 GROUP BY LOGIN
ORDER BY MINI ASC;
```

aura comme résultat:

LOGIN	MINI
alain	9
pitou	14

MAX()

Réciproque de la fonction **MIN**, la fonction **MAX()** renvoie la valeur maximum d'une série. Appliqué à la table *table4* le query suivant:

```
SELECT LOGIN,MAX(SCORE) AS MAXI
FROM table4 GROUP BY LOGIN
ORDER BY MAXI DESC;
```

aura comme résultat:

LOGIN	MAXI
pitou	17
alain	14

SUM()

La fonction **SUM()** calcule la somme d'une série. Si l'on désire à partir de la table *table4* connaître le nombre total de points marqués par chacun des joueurs, le query suivant:

```
SELECT LOGIN,SUM(SCORE) AS TOTAL
FROM table4 GROUP BY LOGIN
ORDER BY TOTAL DESC;
```

aura comme résultat:

LOGIN	MAXI
alain	35
pitou	31

Opérateurs divers

BETWEEN

L'opérateur **BETWEEN** dans une clause **WHERE** permet de sélectionner les lignes dont la valeur d'un champ est inclus entre 2 bornes. **BETWEEN** est equivalent à borne1 <= valeur <= borne2. L'opérateur sait travailler avec des chaines de caractères. En reprenant la table *table4*:

table4		
ID	**LOGIN**	**SCORE**
1	alain	12
2	alain	14
3	alain	9
4	pitou	17
5	pitou	14

Le query suivant:

```
SELECT LOGIN,SCORE FROM table4
WHERE SCORE BETWEEN 14 AND 17
ORDER BY SCORE DESC;
```

aura pour résultat:

LOGIN	SCORE
pitou	17
alain	14
pitou	14

LIKE

L'opérateur **LIKE** permet de réaliser une comparaison partielle. Soit la table *table5*:

table5	
EMAIL	**SCORE**
alain@wanadoo.fr	12
charic@wanadoo.be	9
buzuk@infonie.fr	11
charle@arnaq.com	17
pitou@wanadoo.fr	5
alphonse@free.fr	14

Recherche des emails commençant par 'al':

```
SELECT EMAIL,SCORE
FROM table5
WHERE EMAIL LIKE 'al%';
```

aura comme résultat:

EMAIL	SCORE
alain@wanadoo.fr	12
alphonse@free.fr	14

Le caractère % signifiant "n'importe quel nombre de caractères"

Recherche des emails contenant 'wanadoo':

```
SELECT EMAIL,SCORE
FROM table5
WHERE EMAIL LIKE '%wanadoo%'
ORDER BY EMAIL;
```

aura comme résultat:

EMAIL	SCORE
alain@wanadoo.fr	12
charic@wanadoo.be	9
pitou@wanadoo.fr	5

Recherche des adresses finissant par '.fr':

```
SELECT EMAIL,SCORE
FROM table5
WHERE EMAIL LIKE '%.fr'
ORDER BY EMAIL;
```

aura comme résultat:

EMAIL	SCORE
alain@wanadoo.fr	12
alphonse@free.fr	14
buzuk@infonie.fr	11
pitou@wanadoo.fr	5

La clause LIMIT

Jusqu'à présent le nombre de résultats renvoyés par une requête n'était limité que par les conditions. Dans certains cas, une requête même bien spécifiée peut engendrer un nombre de résultats important, ce qui peut être gênant lors de l'affichage (cas d'une liste par exemple).

Bien que non−portable (PostGreSQL la gère depuis peu) la clause **LIMIT** peut vous simplifier grandement la vie.

La Syntaxe de **LIMIT** est la suivante:

LIMIT *debut, nombre_de_lignes*

En reprenant la table *table5* de la page précédente, le query suivant:

```
SELECT * FROM table5
LIMIT 0, 4;
```

aura pour résultat:

EMAIL	SCORE
alain@wanadoo.fr	12
charic@wanadoo.be	9
buzuk@infonie.fr	11
charle@arnaq.com	17

et le query suivant:

```
SELECT * FROM table5
LIMIT 4, 4;
```

aura pour résultat:

EMAIL	SCORE
pitou@wanadoo.fr	5
alphonse@free.fr	14

La clause HAVING

La clause **HAVING** applique une condition supplémentaire sur un ou plusieurs champs listés dans la commande **SELECT**. Cette clause s'applique juste avant l'envoi du résultat et n'est pas interprétée par l'optimiseur, il faut donc eviter de faire un query juste avec **HAVING** sans restriction sur les conditions.

Soit le query suivant:

```
SELECT EMAIL,MAX(SCORE) AS BESTSCORE
FROM table5 GROUP BY EMAIL
HAVING BESTSCORE > 15;
```

aura pour résultat:

EMAIL	BESTSCORE
charle@arnaq.com	17

SELECT multitables

Jusqu'à présent nous n'avons utilisé la commande **SELECT** qu'avec une seule table. La puissance du langage SQL revêt tout son intérêt lorsque l'on doit manipuler des informations provenant de plusieurs tables et que ces informations sont reliées entre elles de façon cohérente.

Soit les deux tables:

JOUEUR		
ID	**LOGIN**	**LEVEL_ID**
1	alain	1
2	pitou	2
3	manu	4
4	gégé	1

LEVEL	
ID	**NIVEAU**
1	Pilote
2	Capitaine
3	Commandant
4	Amiral

Si nous voulons afficher la liste des joueurs avec le niveau qu'ils ont atteint, nous effectuerons le query suivant:

```
SELECT JOUEUR.LOGIN,LEVEL.NIVEAU
FROM JOUEUR, LEVEL
WHERE JOUEUR.LEVEL_ID=LEVEL.ID;
ORDER BY LEVEL.ID DESC;
```

qui aura pour résultat:

LOGIN	NIVEAU
manu	Amiral
pitou	Capitaine
alain	Pilote
gégé	Pilote

Utilisation de AS

L'utilisation de la syntaxe *table.champ* pour accéder à un champ peut entraîner des problèmes de lisibilité dus à l'écriture de queries un peu longs. En affectant un alias au nom des tables à l'aide de **AS** on peut notablement raccourcir l'écriture du query précédent.

Le query devient:

```
SELECT J.LOGIN,L.NIVEAU
FROM JOUEUR AS J, LEVEL AS L
WHERE J.LEVEL_ID=L.ID;
ORDER BY L.ID DESC;
```

et dont le résultat sera identique au précédent.

Dans le cas que nous venons de voir, la relation entre les tables est dite relation 1 −> 1: à chaque enregistrement de la table *JOUEUR* correspond un enregistrement de la table *LEVEL*.

Nous allons maintenant regarder de plus près le cas des relations 1 −> n, qui signifient qu'à tout enregistrement d'une table A correspondent 1 ou plusieurs enregistrements dans une table B.

Soit la table *SCORE* stockant les résultats des parties de chacun des joueurs. Chaque joueur peut avoir fait une ou plusieurs parties. On sait à quel joueur appartient un score en stockant l'identifiant du joueur concerné (à savoir son *ID* dans la table *JOUEUR*). Cet identifiant sera stocké dans le champ *PLAYER_ID*.

SCORE		
ID	**PLAYER_ID**	**POINTS**
1	3	12
2	2	11
3	1	10
4	3	14
5	2	13
6	3	17
7	4	9

Le query suivant:

```
SELECT J.LOGIN,SUM(S.POINTS) AS TSCORE
FROM JOUEUR AS J, SCORE AS S
WHERE J.ID=S.PLAYER_ID;
ORDER BY TSCORE DESC;
```

aura pour résultat:

LOGIN	TSCORE
manu	43
pitou	24
alain	10
gégé	9

Les exemples multi−tables précédents constituent ce que l'on nomme une jointure croisée. Il suffit de sélectionner les enregistrements en vérifiant l'égalité du contenu d'un champ de la première table et d'un champ de la seconde table.

LEFT JOIN...ON

Nous allons maintenant étudier un cas particulier qui est celui où l'on recherche non pas les valeurs présentes dans une table, mais au contraire des valeurs absentes. La jointure **LEFT JOIN** va nous permettre de réaliser cette opération.

Si l'on reprend la table *LEVEL* et que l'on effectue le query suivant:

```
SELECT NIVEAU,LOGIN FROM LEVEL AS L
LEFT JOIN JOUEUR ON L.ID=JOUEUR.LEVEL_ID;
```

nous aurons comme résultat:

NIVEAU	LOGIN
pilote	alain
pilote	gégé
Capitaine	pitou
Commandant	
Amiral	manu

Nous remarquons que pour le niveau *Commandant*, la colonne *LOGIN* est vide. Si nous désirons répondre à la question 'Quel niveau n'est pas affecté à au moins un joueur?' nous écrivons le query suivant:

```
SELECT NIVEAU FROM LEVEL AS L
LEFT JOIN JOUEUR ON L.ID=JOUEUR.LEVEL_ID
WHERE LOGIN IS NULL;
```

qui aura pour résultat:

NIVEAU
Commandant

KDO kdo@zephpmag.com