Concevez votre site web avec PHP et MySQL

30 hours Medium License

Les fonctions SQL

Vous connaissez déjà les fonctions en PHP, mais vous allez découvrir dans ce chapitre que SQL propose lui aussi toute une série de fonctions ! Le langage SQL permet en effet d'effectuer des calculs directement sur ses données à l'aide de fonctions toutes prêtes.

Celles-ci sont moins nombreuses qu'en PHP mais elles sont spécialement dédiées aux bases de données et se révèlent très puissantes dans la pratique. Pour reprendre notre exemple de la table `jeux_video`, elles permettent de récupérer très simplement le prix moyen de l'ensemble des jeux, de compter le nombre de jeux que possède chaque personne, d'extraire le jeu le plus cher ou le moins cher, etc. Les fonctions se révèlent également indispensables lorsqu'on doit travailler avec des dates en SQL, comme nous le ferons dans le chapitre suivant.

Les fonctions SQL peuvent être classées en deux catégories :

- **les fonctions scalaires** : elles agissent sur chaque entrée. Par exemple, vous pouvez transformer en majuscules la valeur de chacune des entrées d'un champ ;
- **les fonctions d'agrégat** : lorsque vous utilisez ce type de fonctions, des calculs sont faits sur l'ensemble de la table pour retourner **une** valeur. Par exemple, calculer la moyenne des prix retourne une valeur : le prix moyen.

Les fonctions scalaires

Nous allons d'abord découvrir le mode d'emploi d'une fonction SQL de type **scalaire** : la fonction `UPPER`. Lorsque vous aurez appris à vous en servir, vous serez capables de faire de même avec toutes les autres fonctions scalaires. Je vous proposerai alors une petite sélection de fonctions scalaires à connaître, sachant qu'il en existe d'autres mais que nous ne pouvons pas toutes les passer en revue car ce serait bien trop long.

Utiliser une fonction scalaire SQL

Pour nos exemples nous allons nous baser sur la table `jeux_video` que nous connaissons bien maintenant. Pour rappel, voici à quoi elle ressemble :

ID	nom	possesseur	console	prix	nbre_joueurs_max	commentaires
1	Super Mario Bros	Florent	NES	4	1	Un jeu d'anthologie !
2	Sonic	Patrick	Megadrive	2	1	Pour moi, le meilleur jeu au monde !
3	Zelda : ocarina of time	Florent	Nintendo 64	15	1	Un jeu grand, beau et complet comme on en voit rarement de nos jours
4	Mario Kart 64	Florent	Nintendo 64	25	4	Un excellent jeu de kart !
5	Super Smash Bros Melee	Michel	GameCube	55	4	Un jeu de baston délirant !

On écrit les noms des fonctions SQL en majuscules, comme on le fait déjà pour la plupart des mots-clés comme `SELECT`, `INSERT`, etc. Ce n'est pas une obligation mais plutôt une convention, une habitude qu'ont prise les programmeurs.

Pour vous montrer comment on utilise les fonctions scalaires SQL, je vais me baser sur la fonction `UPPER()` qui permet de convertir l'intégralité d'un champ en majuscules. Supposons que nous souhaitions obtenir les noms de tous les jeux en majuscules ; voici comment on écrirait la requête SQL :

```php
SELECT UPPER(nom) FROM jeux_video
```

La fonction `UPPER` est utilisée sur le champ `nom`. On récupère ainsi tous les noms des jeux en majuscules.

Cela modifie-t-il le contenu de la table ?

Non ! La table reste la même. La fonction `UPPER` modifie seulement la valeur envoyée à PHP. On ne touche donc pas au contenu de la table.

Cela crée en fait un « champ virtuel » qui n'existe que le temps de la requête. Il est conseillé de donner un nom à ce champ virtuel qui représente les noms des jeux en majuscules. Il faut utiliser pour cela le mot-clé `AS`, comme ceci :

```php
SELECT UPPER(nom) AS nom_maj FROM jeux_video
```

On récupère les noms des jeux en majuscules via un champ virtuel appelé `nom_maj`.

Ce champ virtuel est appelé **alias**.

Voici le tableau que retournera MySQL après la requête précédente :

nom_maj
SUPER MARIO BROS
SONIC
ZELDA : OCARINA OF TIME
MARIO KART 64
SUPER SMASH BROS MELEE

On peut s'en servir en PHP pour afficher les noms des jeux en majuscules :

```php
<?php
$reponse = $bdd->query('SELECT UPPER(nom) AS nom_maj FROM jeux_video');

while ($donnees = $reponse->fetch())
{
    echo $donnees['nom_maj'] . '<br />';
}
```

```
8
9  $reponse->closeCursor();
10
11 ?>
```

SUPER MARIO BROS
SONIC
ZELDA : OCARINA OF TIME
MARIO KART 64
SUPER SMASH BROS MELEE
DEAD OR ALIVE
DEAD OR ALIVE XTREME BEACH VOLLEY BALL
ENTER THE MATRIX
MAX PAYNE 2
YOSHI'S ISLAND
COMMANDOS 3
FINAL FANTASY X
POKEMON RUBIS
STARCRAFT
GRAND THEFT AUTO 3
HOMEWORLD 2
ALADIN
SUPER MARIO BROS 3
SSX 3
STAR WARS : JEDI OUTCAST
ACTUA SOCCER 3
TIME CRISIS 3
X-FILES

Afficher les noms des jeux en majuscules

Comme vous le voyez, PHP ne récupère qu'un champ nommé `nom_maj` (même s'il n'existe pas dans la table). En affichant le contenu de ce champ, on ne récupère que les noms des jeux en majuscules.

Bien entendu, vous pouvez aussi récupérer le contenu des autres champs comme avant sans forcément leur appliquer une fonction :

php

```
1 SELECT UPPER(nom) AS nom_maj, possesseur, console, prix FROM jeux_video
```

On récupèrera alors les données suivantes :

nom_maj	possesseur	console	prix
SUPER MARIO BROS	Florent	NES	4
SONIC	Patrick	Megadrive	2
ZELDA : OCARINA OF TIME	Florent	Nintendo 64	15
MARIO KART 64	Florent	Nintendo 64	25
SUPER SMASH BROS MELEE	Michel	GameCube	55

Vous savez maintenant utiliser une fonction SQL scalaire. ;-)
Passons en revue quelques fonctions du même type, et qui s'utilisent donc de la même manière.

Présentation de quelques fonctions scalaires utiles

Je vais vous présenter une sélection de fonctions scalaires qu'il peut être utile de connaître. Il en existe bien d'autres comme nous le verrons à la fin de cette liste, mais il serait trop long et peu utile de toutes les présenter ici.

UPPER : convertir en majuscules

Cette fonction convertit le texte d'un champ en majuscules. Nous l'avons découverte pour introduire les fonctions SQL :

```php
SELECT UPPER(nom) AS nom_maj FROM jeux_video
```

Ainsi, le jeu « Sonic » sera renvoyé sous la forme « SONIC » dans un champ nommé `nom_maj`.

LOWER : convertir en minuscules

Cette fonction a l'effet inverse : le contenu sera entièrement écrit en minuscules.

```php
SELECT LOWER(nom) AS nom_min FROM jeux_video
```

Cette fois, le jeu « Sonic » sera renvoyé sous la forme « sonic » dans un champ nommé `nom_min`.

LENGTH : compter le nombre de caractères

Vous pouvez obtenir la longueur d'un champ avec la fonction `LENGTH()` :

```php
SELECT LENGTH(nom) AS longueur_nom FROM jeux_video
```

Pour « Sonic », on récupèrera donc la valeur 5 dans un champ `longueur_nom`.

ROUND : arrondir un nombre décimal

La fonction `ROUND()` s'utilise sur des champs comportant des valeurs décimales. Il n'y en a pas dans la table `jeux_video`, mais si on avait des prix décimaux, on pourrait arrondir les valeurs avec cette fonction.

Celle-ci prend cette fois deux paramètres : le nom du champ à arrondir et le nombre de chiffres après la virgule que l'on souhaite obtenir. Exemple :

```php
SELECT ROUND(prix, 2) AS prix_arrondi FROM jeux_video
```

Ainsi, si un jeu coûte 25,86999 euros, on obtiendra la valeur 25,87 euros dans un champ `prix_arrondi`.

Et bien d'autres !

Il existe beaucoup d'autres fonctions SQL du même type mais je ne peux pas toutes vous les présenter. La documentation de MySQL vous propose une liste bien plus complète de fonctions mathématiques (comme `ROUND`) et de fonctions sur les chaînes de caractères (comme `UPPER`). Si vous voulez en découvrir d'autres, c'est par là qu'il faut aller !

Les fonctions d'agrégat

Comme précédemment, nous allons d'abord voir comment on utilise une fonction d'agrégat dans une requête SQL et comment on récupère le résultat en PHP, puis je vous présenterai une sélection de fonctions à connaître. Bien entendu, il en existe bien d'autres que vous pourrez découvrir dans la documentation. L'essentiel est de comprendre comment s'utilise ce type de fonctions : vous pourrez ensuite appliquer ce que vous connaissez à n'importe quelle autre fonction du même type.

Utiliser une fonction d'agrégat SQL

Ces fonctions diffèrent assez des précédentes. Plutôt que de modifier des valeurs une à une, elles font des opérations sur plusieurs entrées pour retourner une seule valeur.

Par exemple, `ROUND` permettait d'arrondir chaque prix. On récupérait autant d'entrées qu'il y en avait dans la table. En revanche, une fonction d'agrégat comme `AVG` renvoie **une seule entrée** : la valeur moyenne de tous les prix.

Regardons de près la fonction d'agrégat `AVG`. Elle calcule la moyenne d'un champ contenant des nombres. Utilisons-la sur le champ `prix` :

```php
SELECT AVG(prix) AS prix_moyen FROM jeux_video
```

On donne là encore un alias au résultat donné par la fonction. La particularité, c'est que cette requête ne va retourner qu'une seule entrée, à savoir le prix moyen de tous les jeux :

prix_moyen
28.34

Pour afficher cette information en PHP, on pourrait faire comme on en a l'habitude (cela fonctionne) :

```php
<?php
$reponse = $bdd->query('SELECT AVG(prix) AS prix_moyen FROM jeux_video');

while ($donnees = $reponse->fetch())
{
    echo $donnees['prix_moyen'];
```

```
}

$reponse->closeCursor();

?>
```

Néanmoins, pourquoi s'embêterait-on à faire une boucle étant donné qu'on **sait** qu'on ne va récupérer qu'une seule entrée, puisqu'on utilise une fonction d'agrégat ?

On peut se permettre d'appeler `fetch()` une seule fois et en dehors d'une boucle étant donné qu'il n'y a qu'une seule entrée. Le code suivant est donc un peu plus adapté dans le cas présent :

php

```php
<?php
$reponse = $bdd->query('SELECT AVG(prix) AS prix_moyen FROM jeux_video');

$donnees = $reponse->fetch();
echo $donnees['prix_moyen'];

$reponse->closeCursor();

?>
```

Ce code est plus simple et plus logique. On récupère la première et seule entrée avec `fetch()` et on affiche ce qu'elle contient, puis on ferme le curseur. Inutile de le faire dans une boucle étant donné qu'il n'y a pas de seconde entrée.

N'hésitez pas à filtrer !

Bien entendu, vous pouvez profiter de toute la puissance du langage SQL pour obtenir, par exemple, le prix moyen des jeux appartenant à Patrick. Voici comment on s'y prendrait :

php

```php
SELECT AVG(prix) AS prix_moyen FROM jeux_video WHERE possesseur='Patrick'
```

Le calcul de la moyenne ne sera fait que sur la liste des jeux qui appartiennent à Patrick. Vous pourriez même combiner les conditions pour obtenir le prix moyen des jeux de Patrick qui se jouent à un seul joueur. Essayez !

Ne pas mélanger une fonction d'agrégat avec d'autres champs

Soyez attentifs à ce point car il n'est pas forcément évident à comprendre : vous **ne devez pas** récupérer d'autres champs de la table quand vous utilisez une fonction d'agrégat, contrairement à tout à l'heure avec les fonctions scalaires. En effet, quel sens cela aurait-il de faire :

php

```php
SELECT AVG(prix) AS prix_moyen, nom FROM jeux_video
```

On récupérerait d'un côté le prix moyen de tous les jeux et de l'autre la liste des noms de tous les jeux… Il est impossible de représenter ceci dans un seul et même tableau.

Comme vous le savez, SQL renvoie les informations sous la forme d'un tableau. Or on ne peut pas représenter la moyenne des prix (qui tient en une seule entrée) en même temps que la liste des jeux. Si on voulait obtenir ces deux informations il faudrait faire deux requêtes.

Présentation de quelques fonctions d'agrégat utiles

AVG : calculer la moyenne

C'est la fonction que l'on vient d'étudier pour découvrir les fonctions d'agrégat. Elle retourne la moyenne d'un champ contenant des nombres :

```php
SELECT AVG(prix) AS prix_moyen FROM jeux_video
```

SUM : additionner les valeurs

La fonction `SUM` permet d'additionner toutes les valeurs d'un champ. Ainsi, on pourrait connaître la valeur totale des jeux appartenant à Patrick :

```php
SELECT SUM(prix) AS prix_total FROM jeux_video WHERE possesseur='Patrick'
```

MAX : retourner la valeur maximale

Cette fonction analyse un champ et retourne la valeur maximale trouvée. Pour obtenir le prix du jeu le plus cher :

```php
SELECT MAX(prix) AS prix_max FROM jeux_video
```

MIN : retourner la valeur minimale

De même, on peut obtenir le prix du jeu le moins cher :

```php
SELECT MIN(prix) AS prix_min FROM jeux_video
```

COUNT : compter le nombre d'entrées

La fonction `COUNT` permet de compter le nombre d'entrées. Elle est très intéressante mais plus complexe. On peut en effet l'utiliser de plusieurs façons différentes.

L'utilisation la plus courante consiste à lui donner `*` en paramètre :

```php
SELECT COUNT(*) AS nbjeux FROM jeux_video
```

On obtient ainsi le nombre total de jeux dans la table.

On peut bien entendu filtrer avec une clause `WHERE`, pour obtenir le nombre de jeux appartenant à Florent par exemple :

```php
SELECT COUNT(*) AS nbjeux FROM jeux_video WHERE possesseur='Florent'
```

Il est possible de compter uniquement les entrées pour lesquelles l'un des champs n'est pas vide, c'est-à-dire qu'il ne vaut pas `NULL`. Il n'y a pas de jeu de ce type dans notre table `jeux_video`, mais supposons que pour certains jeux on ne connaisse pas le nombre de joueurs maximum. On laisserait certaines entrées vides, ce qui aurait pour effet d'afficher `NULL` (pas de valeur) dans la colonne `nbre_joueurs_max` (comme dans le tableau suivant).

ID	nom	possesseur	console	prix	nbre_joueurs_max	commentaires
1	Super Mario Bros	Florent	NES	4	NULL	Un jeu d'anthologie !

ID	nom	possesseur	console	prix	nbre_joueurs_max	commentaires
2	Sonic	Patrick	Megadrive	2	NULL	Pour moi, le meilleur jeu au monde !
3	Zelda : ocarina of time	Florent	Nintendo 64	15	1	Un jeu grand, beau et complet comme on en voit rarement de nos jours
4	Mario Kart 64	Florent	Nintendo 64	25	4	Un excellent jeu de kart !
5	Super Smash Bros Melee	Michel	GameCube	55	NULL	Un jeu de baston délirant !

Dans ce cas, on peut compter uniquement les jeux qui ont un nombre de joueurs maximum défini. On doit indiquer en paramètre le nom du champ à analyser :

```php
SELECT COUNT(nbre_joueurs_max) AS nbjeux FROM jeux_video
```

Dans notre exemple, seuls les jeux *Zelda* et *Mario Kart* seront comptés car on connaît leur nombre de joueurs maximum. Donc on obtiendra « 2 » en réponse.

Enfin, il est possible de compter le nombre de valeurs distinctes sur un champ précis. Par exemple dans la colonne `possesseur`, Florent apparaît plusieurs fois, Patrick aussi, etc. Mais combien y a-t-il de personnes différentes dans la table ? On peut le savoir en utilisant le mot-clé `DISTINCT` devant le nom du champ à analyser, comme ceci :

```php
SELECT COUNT(DISTINCT possesseur) AS nbpossesseurs FROM jeux_video
```

On peut ainsi facilement savoir combien de personnes différentes sont référencées dans la table. Essayez de faire de même pour connaître le nombre de consoles différentes dans la table !

GROUP BY et HAVING : le groupement de données

Je vous disais un peu plus tôt qu'on ne pouvait pas récupérer d'autres champs lorsqu'on utilisait une fonction d'agrégat. Prenons par exemple la requête suivante :

```php
SELECT AVG(prix) AS prix_moyen, console FROM jeux_video
```

Ça n'a pas de sens de récupérer le prix moyen de tous les jeux et le champ « console » à la fois. Il n'y a qu'un seul prix moyen pour tous les jeux, mais plusieurs consoles. MySQL ne peut pas renvoyer un tableau correspondant à ces informations-là.

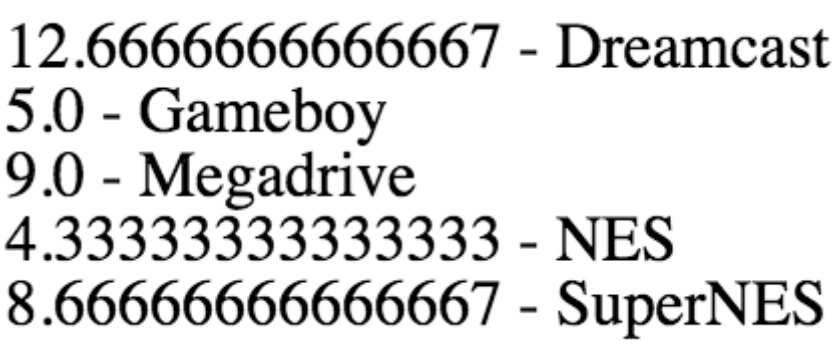
GROUP BY : grouper des données

En revanche, ce qui pourrait avoir du sens, ce serait de demander le **prix moyen des jeux pour chaque console** ! Pour faire cela, on doit utiliser un nouveau mot-clé : `GROUP BY`. Cela signifie « grouper par ». On utilise cette clause en

combinaison d'une fonction d'agrégat (comme `AVG`) pour obtenir des informations intéressantes sur des groupes de données.

Voici un exemple d'utilisation de `GROUP BY` :

```php
SELECT AVG(prix) AS prix_moyen, console FROM jeux_video GROUP BY console
```



GROUP BY console

Il faut utiliser `GROUP BY` en même temps qu'une fonction d'agrégat, sinon il ne sert à rien. Ici, on récupère le prix moyen et la console, et on choisit de grouper par console. Par conséquent, on obtiendra la liste des différentes consoles de la table et le prix moyen des jeux de chaque plate-forme !

prix_moyen	console
12.67	Dreamcast
5.00	Gameboy
47.50	GameCube

Cette fois les valeurs sont cohérentes ! On a la liste des consoles et le prix moyen des jeux associés.

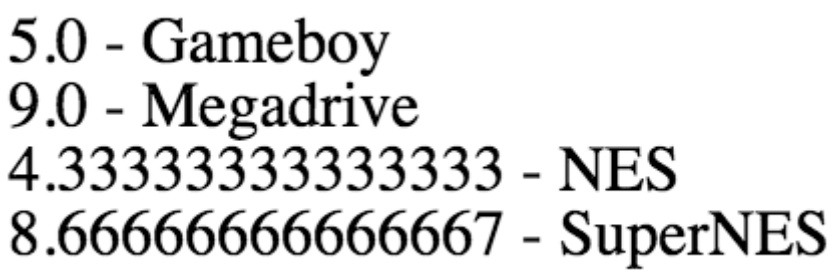
Exercice : essayez d'obtenir de la même façon la valeur totale des jeux que possède chaque personne.

HAVING : filtrer les données regroupées

`HAVING` est un peu l'équivalent de `WHERE`, mais il agit sur les données une fois qu'elles ont été regroupées. C'est donc une façon de filtrer les données à la fin des opérations.

Voyez la requête suivante :

```php
SELECT AVG(prix) AS prix_moyen, console FROM jeux_video GROUP BY console HAVING prix_moyen <= 10
```



HAVING prix_moyen <= 10

Avec cette requête, on récupère uniquement la liste des consoles et leur prix moyen si ce prix moyen ne dépasse pas 10 euros.

`HAVING` ne doit s'utiliser que sur le résultat d'une fonction d'agrégat. Voilà pourquoi on l'utilise ici sur `prix_moyen` et non sur `console`.

Je ne comprends pas la différence entre `WHERE` et `HAVING`. Les deux permettent de filtrer, non ?

Oui, mais pas au même moment. `WHERE` agit en premier, avant le groupement des données, tandis que `HAVING` agit en second, après le groupement des données. On peut d'ailleurs très bien combiner les deux, regardez l'exemple suivant :

```php
SELECT AVG(prix) AS prix_moyen, console FROM jeux_video WHERE possesseur='Patrick' GROUP BY console HAVING prix_moyen <= 10
```

WHERE agit avant HAVING

Ça commence à faire de la requête maousse costaude. ;-)

Ici, on demande à récupérer le prix moyen par console de tous les jeux de Patrick (`WHERE`), à condition que le prix moyen des jeux de la console ne dépasse pas 10 euros (`HAVING`).

En résumé

- MySQL permet d'exécuter certaines fonctions lui-même, sans avoir à passer par PHP. Ces fonctions modifient les données renvoyées.
- Il existe deux types de fonctions :
  - les **fonctions scalaires** : elles agissent sur chaque entrée récupérée. Elles permettent par exemple de convertir tout le contenu d'un champ en majuscules ou d'arrondir chacune des valeurs ;
  - les **fonctions d'agrégat** : elles effectuent des calculs sur plusieurs entrées pour retourner une et une seule valeur. Par exemple : calcul de la moyenne, somme des valeurs, comptage du nombre d'entrées…

- On peut donner un autre nom aux champs modifiés par les fonctions en créant des alias à l'aide du mot-clé `AS`.
- Lorsqu'on utilise une fonction d'agrégat, il est possible de regrouper des données avec `GROUP BY`.
- Après un groupement de données, on peut filtrer le résultat avec `HAVING`. Il ne faut pas le confondre avec WHERE qui filtre avant le groupement des données.

TP : un mini-chat

Les dates en SQL

The author

Mathieu Nebra

Entrepreneur à plein temps, auteur à plein temps et co-fondateur d'OpenClassrooms :o)

Check out this course via