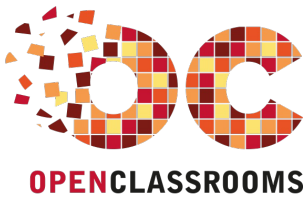
Introduction aux opérateurs de bits

Par infotoubib



www.openclassrooms.com

Dernière mise à jour le 6/01/2011

Sommaire

Introduction aux opérateurs de bits

Par  infotoubib

Mise à jour : 06/01/2011

Vous souhaitez créer un système de droits d'accès pour votre site, vous utilisez des objets aux multiples propriétés, vous voulez passer de nombreux paramètres de type oui / non à une seule fonction ?
Les opérateurs de bits peuvent vous aider !

Ne connaissant pas le niveau de chacun, j'ai beaucoup détaillé. Mais une fois que l'on a compris le raisonnement, l'utilisation de ces opérateurs est très simple !

Sommaire du tutoriel :

Vous avez dit opérateur ?

Quel que soit le programme que vous créerez, vous allez utiliser des opérateurs. Un opérateur, c'est ce qui vous permet d'obtenir une valeur à partir d'une ou de plusieurs autres valeurs. Il en existe différentes catégories :

- certaines que vous connaissez probablement :

- opérateurs arithmétiques (+ , - , / , * ,...),
- opérateurs de comparaison (== , > , < ,...),
- opérateurs logiques (**&&** , || , **!** ,...),
- opérateurs d'assignation (= , .= , += ,...).

- d'autres semblent plus obscurs, dont les opérateurs de bits. Nous allons utiliser une partie de ces derniers.

Pour une liste exhaustive des opérateurs, voir :
- opérateurs du PHP ;
- opérateurs du C++ (en anglais).

Gérer les droits des utilisateurs

Supposons que vous vouliez instaurer un système de droits pour vos pages. Certains des membres auront le droit de publier un article, d'autres seulement de le proposer, d'autres seront restreints à la lecture. D'autres pages du site seront réservées aux administrateurs ou au webmaster... Supposons un site complexe dans lequel une dizaine de droits puissent être attribués (ou non) aux membres...

La gestion des droits peut être simplifiée en général en les distribuant sur quelques groupes. Pour notre exemple, nous allons les détailler, ce qui est parfois utile...

Une première possibilité serait de créer, dans la table membres, un champ par droit attribuable. Lors de la connexion du membre,

les droits sont placés dans des variables de session.

Code : PHP

```php
<?php
        // Connexion à la base de données [...]
        $Resultat = mysql_query("SELECT * FROM membres WHERE
Pseudo='$Pseudo'");
        if ($Donnee = mysql_fetch_array($Resultat)) {
                //... Vérifier le mot de passe, etc. ...
                if ($Donnee['lire_article'])
$_SESSION['droits']['lire_article'] = TRUE;
                if ($Donnee['ecrire_article'])
$_SESSION['droits']['ecrire_article'] = TRUE;
                if ($Donnee['publier_article'])
$_SESSION['droits']['publier_article'] = TRUE;
                if ($Donnee['proposer_new'])
$_SESSION['droits']['proposer_new'] = TRUE;
                if ($Donnee['valider_new'])
$_SESSION['droits']['valider_new'] = TRUE;
                if ($Donnee['gerer_membres'])
$_SESSION['droits']['gerer_membres'] = TRUE;
                if ($Donnee['gerer_base'])
$_SESSION['droits']['gerer_base'] = TRUE;
                // etc.
        }
?>
```

Lorsqu'une page doit être restreinte, on teste :

Code : PHP

```php
<?php
        /* Le membre a-t-il droit à une page donnée ? */
        if (empty ($_SESSION['droits']['proposer_new'])) {
                echo 'Tu n\'as rien à faire ici !!';
                exit;
        }
?>
```

Cette première solution, que je vous déconseille vivement, fonctionnerait avec un petit nombre de droits. Avec dix, c'est vraiment lourd — aussi bien pour le programmeur que pour le serveur, qui devra garder des tas de variables en mémoire, et faire de nombreuses opérations... 

Nous allons donc placer l'ensemble des droits du membre dans une seule variable. Là encore, plusieurs possibilités s'offrent à nous.

Une première idée pourrait être de symboliser chaque droit par une lettre et de tester si la lettre requise pour autoriser la page se trouve bien dans la variable Droits du membre. Ce qui donnerait :

Code : PHP

```php
<?php
        /* les droits (la correspondance sera conservée dans une
table par exemple)
A : lire un article
B : écrire un article
C : publier un article
etc. */

        /* Pour attribuer les droits, on utilisera l'opérateur de
concaténation */
        $DroitsDuMembre = '';
```

```
            if ($Droit_lecture)     $DroitsDuMembre .= 'A';
            if ($Droit_Publication) $DroitsDuMembre .= 'C';
            // etc.

            /* Lors du login de l'utilisateur */
            // Connexion à la base de données [...]

            $Resultat = mysql_query("SELECT Nom, Pass, Droits FROM
    membres WHERE Pseudo='$Pseudo'");
            if ($Donnee = mysql_fetch_array($Resultat)) {
                  //... Vérifier le mot de passe, etc ...
                  $_SESSION['droits'] = $Donnee['Droits'];
            }


            /* Le membre a-t-il droit à une page donnée ? */
            if (strpos($_SESSION['droits'], 'C') === FALSE) {
                  echo 'Désolé, mais cette page t\'est interdite...';
                  exit;
            }
    ?>
```

C'est déjà nettement mieux !

Il existe cependant une solution encore plus économique pour l'ordinateur : penser binaire !

Penser binaire

Pour l'ordinateur, un nombre est une suite de 0 et de 1, chacun correspondant à un bit. La plupart de nos ordinateurs travaillent par séries de 32 bits.

Sur un nombre, nous pouvons donc avoir 32 bits pouvant prendre chacun la valeur **1** (qui signifiera pour nous : « vrai » ou « oui ») ou **0** (qui signifiera bien sûr « faux » ou « non »). C'est donc largement suffisant pour stocker notre table de droits !

En fait, en PHP, nous ne pouvons utiliser que 31 bits, car nous ne pouvons pas maîtriser la façon dont les nombres sont stockés, et que le 32e bit (le plus à gauche) peut aussi bien signifier un nombre négatif que 2147483648 (2 puissance 31).
Ce problème ne se pose pas en C/C++ puisque les variables ont un type bien défini.

Nous pouvons décider que (les nombres sont ici sur 16 bits) :
0000000000000001 signifie "lire un article"
0000000000000010 signifie "écrire un article"
0000000000000100 signifie "publier un article"
0000000000001000 signifie "proposer une new"
0000000000010000 signifie "valider une new"
0000000000100000 signifie "gérer les membres"
0000000001000000 signifie "gérer la base de données"

Pour dire que Untel a le droit de lire, écrire, publier des articles, et gérer les membres, il suffira d'attribuer la valeur suivante à sa variable $Droit :
0000000000100111

Pour savoir si Untel a le droit d'écrire un article, il suffit de regarder si le deuxième bit à partir de la droite est bien un **1**.

Pour lui ajouter le droit de sauvegarder la base de données du site et de télécharger la sauvegarde, nous modifierons simplement le septième bit à partir de la droite en **1**.

Pour lui retirer ce droit (il vaut mieux ne pas laisser la base de données entre les mains de n'importe qui...), nous donnerons à nouveau à ce même septième bit la valeur **0**.

Ces opérations sur des bits sont très simples et rapides pour l'ordinateur.

Mais comment effectuer ces opérations ? Nous n'allons quand même pas avoir à écrire en base 2 (0000000000100111) dans le programme ?!

Les opérateurs de bits sont là pour nous faciliter la tâche. Il est donc temps de les découvrir. Nous en verrons quatre dans ce tuto :

& : ET binaire
| : OU binaire (au sens de ET / OU)
^ : XOR binaire (OU exclusif)
~ : NON binaire

Puis nous verrons comment faire en pratique pour ne pas avoir à écrire en binaire.

[Edit] Finalement, nous introduirons aussi les opérateurs de décalage << et >> .

Les opérateurs de bits & et |

Rappelez-vous que pour PHP (comme en C / C++), si un résultat est égal à 0, il est FAUX. Dans le cas contraire, il est VRAI.

Untel a-t-il le droit d'écrire un article ? Opérateur &

Le ET binaire compare deux nombres binaires et donne un résultat dans lequel seuls les bits **1** en même position dans les deux nombres gardent une valeur **1**. Tous les autres sont mis à zéro.

Il faut que le bit n du premier nombre ET du deuxième nombre soient à **1** pour que le bit n du résultat soit à **1**.

0000000000100111 Droits de Untel
& opérateur ET
0000000000000010 écrire un article
=
0000000000000010 Résultat qui est différent de 0000000000000000 donc VRAI

Ajouter le droit de sauvegarder la base de données : Opérateur |

Le OU binaire compare deux nombres binaires et donne un résultat dans lequel les bits **1** de chacun des deux nombres gardent une valeur **1**.

Il suffit que le bit n du premier nombre ET / OU du deuxième nombre soit à **1** pour que le bit n du résultat soit à **1**.

0000000000100111 Droits de Untel
| opérateur OU
0000000001000000 gérer la base
=
0000000001100111 qui est le nouveau droit de Untel

Observation : le résultat aurait été le même si Untel disposait déjà de ce droit.
0000000001100111
|
0000000001000000
=
0000000001100111

Les opérateurs de bits ^ et ~

Retirer le droit de sauvegarder la base de données : ~ et ^

Selon que nous connaissons ou non les droits actuels de Untel, nous pourrons utiliser un opérateur (^) ou une combinaison de deux opérateurs (& et ~).

Une méthode risquée pour retirer un droit : le OU Exclusif

Le OU Exclusif binaire ne retient que les **1** présents dans UN SEUL des deux nombres examinés.

Il faut que le bit n du premier nombre OU du deuxième nombre, mais pas des deux, soit à **1** pour que le bit n du résultat soit à **1**.

0000000001100111 Droits de Untel
^ opérateur OU Exclusif (XOR)
0000000001000000 gérer la base
=
0000000000100111 qui est le nouveau droit de Untel

Si nous utilisions ^ sans vérifier si Untel avait bien le droit que nous voulions supprimer, et qu'il ne l'ait pas, le même opérateur le lui donnerait : en fait, on peut utiliser plutôt cet opérateur pour inverser un droit.
0000000000100111 Droits de Untel
^ opérateur OU Exclusif (XOR)
0000000001000000 gérer la base
=
0000000001100111 qui est le nouveau droit de Untel

Meilleure méthode : utiliser un masque NON ~

L'opérateur NON binaire inverse tous les bits du nombre auquel il est appliqué.

Chaque **1** deviendra un **0** et chaque **0** deviendra un **1**.

Cela nous permet de créer un masque que nous appliquerons aux droits de Untel pour supprimer uniquement le droit de gestion de la base.
~ 0000000001000000 gérer la base
= 1111111110111111 masque du droit de gestion de la base
& 0000000001100111 droits de Untel
= 0000000000100111 nouveaux droits de Untel

On peut observer qu'appliquer le même masque si Untel ne dispose pas du droit à supprimer ne l'ajoutera pas :
1111111110111111 masque du droit de gestion de la base
& 0000000000100111 droits de Untel
= 0000000000100111 nouveaux droits de Untel

Comparer les droits de deux utilisateurs : encore l'opérateur ^

On peut utiliser l'opérateur logique ==, mais l'opérateur de bits ^ est un autre moyen. En fait, le résultat peut être interprété comme « est différent de », ce qui permet de déduire l'égalité bit à bit.
0000000000100111 Droits de Untel
^ opérateur XOR (OU exclusif)
0000000000100111 Droits de User2
=
0000000000000000 Résultat qui est = à 0 donc FAUX : il n'y a pas de différence entre les deux utilisateurs.

Au contraire :
0000000000100111 Droits de Untel
^ opérateur XOR (OU exclusif)
0000000000000011 Droits de User2
=

0000000000100100Résultat qui est différent de 0 donc VRAI : les droits des deux utilisateurs sont différents !

Les droits avec les opérateurs de bits

L'hexadécimal à notre rescousse

Nous ne pouvons pas vraiment écrire directement en base 2 en PHP. Ce serait d'ailleurs trop fatigant. Nous pouvons bien sûr écrire en base 10, comme d'habitude. Mais observons le tableau suivant :

Les puissances de 2

base 2	base 10 (décimal)	base 16 (hexadécimal)
0000000000000001	1	0x01
0000000000000010	2	0x02
0000000000000100	4	0x04
0000000000001000	8	0x08
0000000000010000	16	0x10
0000000000100000	32	0x20
0000000001000000	64	0x40
0000000010000000	128	0x80
0000000100000000	256	0x100
0000001000000000	512	0x200
0000010000000000	1024	0x400
0000100000000000	2048	0x800
0001000000000000	4096	0x1000
0010000000000000	8192	0x2000
0100000000000000	16384	0x4000
1000000000000000	32768	0x8000

En base 2, les puissances de 2 s'écrivent avec un seul **1**, tout le reste étant des **0**. Ce sont donc les puissances de 2 qui nous intéressent.

Nous remarquons qu'il est beaucoup plus simple de les écrire en hexadécimal qu'en base 10 ! Il suffit de se souvenir de la progression 1 - 2 - 4 - 8, puis de recommencer en ajoutant un zéro à droite !

En pratique... et en PHP

Voici un petit exemple de ce que pourrait être la gestion des droits avec les opérateurs de bits.

Code : PHP

```php
<?php
    /* Définition des droits dans un fichier PHP *
*******************************************
Les droits pourraient aussi être définis dans une table, ce qui
permettrait facilement d'accroître la liste et de créer des
checkbox dans les formulaires d'inscription. */

    define ('LIRE ARTICLE',    0x01);
```

```
    define ('ECRIRE_ARTICLE',   0x02);
    define ('PUBLIER_ARTICLE',  0x04);
    define ('PROPOSER_NEW',     0x08);
    define ('VALIDER_NEW',      0x10);
    define ('GERER_MEMBRES',    0x20);


    /* Attribution des droits *
************************
On suppose que les variables $Droit_lecture, $Droit_Publication...
sont récupérées d'un formulaire ; on utilise donc l'opérateur OU
Binaire combiné à l'opérateur d'assignation = . */

    $DroitsDuMembre = 0;
    if ($Droit_lecture)     $DroitsDuMembre |= LIRE_ARTICLE;
    if ($Droit_Publication) $DroitsDuMembre |= ECRIRE_ARTICLE;
    // etc ...

    // Le membre devient newser... On lui donne plusieurs droits
d'un coup.
    $DroitsDuMembre |= PROPOSER_NEW | VALIDER_NEW;


    /* Suppression de droits *
**********************/
    // Les news n'étaient pas bonnes ; retrait des droits
correspondants.
    $DroitsDuMembre &= ~(PROPOSER_NEW | VALIDER_NEW);
    // Interprétation de droite à gauche :
    // On a commencé par créer le droit réunissant les deux à
supprimer par | .
    // Ensuite, on a créé le masque de cet ensemble de droits par ~
.
    // Enfin, on utilise & pour ne garder que les autres droits du
membre.


    /* Login de l'utilisateur *
************************/
    // Connexion à la base de données [...]

    $Resultat = mysql_query("SELECT Nom, Pass, Droits FROM membres
WHERE Pseudo='$Pseudo' ");
    if ($Donnee = mysql_fetch_array($Resultat)) {
        //... Vérifier le mot de passe, etc. ...
        $_SESSION['droits'] = (int)$Donnee['Droits'];
    }


    /* Examen des droits avant d'autoriser l'accès à une page *
*****************************************************/

    /* ATTENTION, les variables de PHP n'ont pas vraiment un type
défini, il est donc utile de signaler au programme que l'on veut
comparer deux nombres et non deux chaînes de caractères. */
    // Le membre a-t-il le droit de publier les articles ?

    // On utilise l'opérateur & pour tester le droit.
    if (!((int)$_SESSION['droits'] & PUBLIER_ARTICLE)) {
        echo 'Désolé, mais cette page t\'est interdite...';
        exit;
    }
?>
```

Appendice : les opérateurs de décalage

Initialement, je ne pensais pas vous en parler, car je ne les avais pas utilisés en PHP. Mais DHKold a suggéré de leur consacrer un petit paragraphe, ce qui me convient tout à fait puisqu'en avançant dans mon projet, j'ai découvert qu'ils étaient très utiles !

Votre site s'est bien développé, et vous proposez à vos membres toute une série d'options (recevoir un mail lorsque l'on a un message personnel, afficher ou masquer tel ou tel menu...). Pour ne pas encombrer votre table « membres », vous décidez d'utiliser un champ (entier non signé) pour stocker l'ensemble des options. Toutes les options sont rassemblées dans une table de votre base de données, et vous voulez créer un script pour gérer cette table directement en ligne.

Lister les puissances de 2 : opérateur << (décalage à gauche)

À chaque option doit correspondre un ID qui soit une puissance de 2 (un nombre avec un seul bit à 1). Mais comment vérifier que l'entrée saisie dans le champ ID en soit une ?

Puisqu'il n'y a que trente-et-une puissances de 2 utilisables, nous allons les tester toutes ! 

L'algorithme pourrait s'écrire :

Code : Autre

```
Pour la suite des puissances de 2 $p2 à partir de 1 (= 2 puissance 0), et inférie

        si $p2 est égal au nombre testé : le nombre testé est une puissance de 2

        sinon je multiplie $p2 par 2 et je recommence le test....
```

Ce sera peut-être plus clair en PHP :

Code : PHP

```
function EstPuissanceDe2 ($Nombre_a_tester) {
        for ($i = 1 ; $i < 0x80000000 ; $i *= 2) {
                if ($i == $Nombre_a_tester) {
                        // Le nombre est une puissance de 2
                        return TRUE;
                }
        }
        // Le nombre n'est pas une puissance de 2 < 2 puissance 31
        return FALSE;
}
```

N'utilisez pas le code ci-dessus : il n'est pas très efficace, et en fait, ce n'est pas sûr qu'il fonctionne !

Nous avons vu que pour multiplier un nombre écrit en binaire par 2, il suffisait de rajouter un 0 à droite. C'est exactement ce que fait l'opérateur de décalage vers la gauche, que l'on note <<. Il rajoute de un à trente-deux zéro(s) à droite (et supprime les un à trente-deux bits les plus à gauches).

0000000000001010 << 1 est égal à 0000000000010100 (multiplication par 2)
0000000000001010 << 4 est égal à 0000000010100000 (multiplication par 2 puissance 4 = 16)

Pour l'ordinateur, lui demander de faire un décalage d'un bit à gauche est encore plus simple à comprendre que de lui demander de multiplier par 2.

Nous pouvons donc remplacer l'instruction $i * 2 par $i << 1

Reste un petit problème :
2 puissance 31 s'écrit en hexadécimal 0x80000000, ce qui est un nombre positif.
En binaire, ce nombre donne : 10000000 00000000 00000000 00000000).
Pour les systèmes 32 bits (la majorité des ordinateurs personnels actuels), ceci peut désigner aussi bien un entier non signé

positif, qu'un entier signé négatif.
Or, comme nous n'avons rien spécifié de particulier, la variable $i sera supposée être un entier signé... Lorsqu'elle prendra la valeur 2 puissance 31, elle sera en fait un nombre négatif, donc inférieure à 0x80000000 => La boucle ne s'interrompt pas !
La prochaine itération donnera donc :
0x80000000 << 1 = 0x00000000 = 0; (Le bit le plus à gauche qui était à 1 est supprimé et on ajoute un 0 à droite : il ne reste que 32 0 !).
Les itérations suivantes donneront toutes le même 0 (appliquer un décalage sur 0 donne toujours 0 par définition). Nous entrerons donc dans une boucle infinie !

La solution sera d'utiliser un des opérateurs de bits que nous avons déjà vu, et de remplacer l'instruction "$i < 0x80000000"; par "$i ^ 0x80000000"; que l'on pourra traduire « tant que $i est différent de 0x80000000 ».

Nous obtenons donc le code suivant :

Code : PHP

```
function EstPuissanceDe2 ($Nombre_a_tester) {
        for ($i = 1 ; $i ^ 0x80000000 ; $i <<= 1) {
                if (! ($i ^ (int) $Nombre_a_tester)) {
                        // s'ils ne sont pas différents, c'est
qu'ils sont égaux !
                        return TRUE;
                }
        }
        // Le nombre n'est pas une puissance de 2 inférieure à 2
puissance 31
        return FALSE;
}
```

Opérateur de décalage vers la droite : >>

C'est, bien sûr, exactement le contraire du décalage vers la gauche : on efface n bit(s) le(s) plus à droite et on ajoute n 0 à gauche, n pouvant aller de 1 à 31 (un décalage à droite ou à gauche de 32 bits ne servirait à rien : le résultat serait toujours égal à 0 sur un système 32 bits !).

Vous voulez afficher un tableau comportant pour chaque membre sa situation par rapport à toutes les options possibles.

Membre	Option 1	Option 2	Option 3	Option 4	Option 5
Titi	oui	non	oui	oui	non
GrosMinet	oui	oui	oui	non	non

Pour remplir les titres du tableau, il suffit de faire une requête SELECT à partir de la table des options. Mais comment afficher les options de chaque membre, puisque ces options sont stockées dans un entier ?
Une façon simple est d'examiner un par un les bits du champ Options des membres, et une petite boucle nous permettra de le faire sans difficulté :

Code : PHP

```
<?php

        // Connexion à la base de données [etc ...]

        // On compte les options.
        $Resultat = mysql_query("SELECT COUNT(*) FROM Options");
        if ($Donnee = mysql_fetch_row($Resultat)) {
                $Nbre_Options = $Donnee[0];
        }

        // On récupère le nom des options et on les liste dans
l'en-tête du tableau.
        // [...] vous savez très bien faire ce code...
```

```
        // Afficher les options pour chaque membre :
        $Resultat = mysql_query("SELECT Nom, Options FROM membres");
        while ($Donnee = mysql_fetch_array($Resultat)) {
                // Pour chaque membre, on affiche le nom…
                echo '<tr><td>', $Donnee['Nom'], '</td>';
                // … et on récupère les options.
                $Options = (int) $Donnee['Options'];
/*******************************
* Partie intéressante du code *
*******************************/
                for ($i = 0 ; $i < $Nbre_Options ; $i ++) {
                        // On examine le bit le plus à droite (1
est le masque correspondant au bit le plus à droite).
                        if ($Options & 1) {
                                echo '<td>OUI</td>';
                        }
                        else {
                                echo '<td>non</td>';
                        }
                        // On passe au bit suivant des Options par
un décalage à droite d'un bit.
                        $Options >>= 1;
                }
/*********************************
* Fin de la partie intéressante *
*********************************/
                // On ferme la ligne.
                echo '</tr>';
        }
        // On ferme la table.
        echo '</table>';
?>
```

Q.C.M.

Le premier QCM de ce cours vous est offert en libre accès.
Pour accéder aux suivants

Connectez-vous Inscrivez-vous

1 - Comment note-t-on l'opérateur de bits OU (et / ou) ?

- ◯ A : ||
- ◯ B : |
- ◯ C : or
- ◯ D : ^
- ◯ E : ^^

2 - Que fait l'opérateur **&** ?
Dans les réponses, j'appelle **1** les bits de valeur 1 et **0** ceux de valeur 0.

- ◯ A : Il retient tous les 1 du premier nombre et tous les 0 du deuxième nombre.
- ◯ B : Il retient tous les 1 du premier nombre et tous les 1 du deuxième nombre.
- ◯ C : Il retient seulement les 0 que les deux nombres ont en commun.
- ◯ D : Il retient seulement les 1 que les deux nombres ont en commun.

3 - Quel sera le résultat de la suite d'opérations suivante ?
Resultat = 00000000 | 00001111 | 00001100

(Attention, j'écris les nombres en base 2 pour que ce soit plus facile :
en PHP, il faudrait plutôt écrire $Resultat = 0x0 | 0xF | 0xC ;).

- ◯ A : Indéfini.
- ◯ B : 00000000 (0).

- ◯ C : 00001100 (0xC).
- ◯ D : 00001111 (0xF).
- ◯ E : 00000011 (0x3).

4 - Soit une fonction WindowsCreate(param) qui reçoit un nombre entier en paramètre pour signifier une série de propriétés :

Code : PHP

```
define ('MAXIMISABLE', 0x1);
        define ('MINIMISABLE', 0x2);
        define ('RESIZABLE',   0x4);
        define ('ICONISABLE',  0x8);
```

Comment puis-je demander une fenêtre qui ait ces quatre propriétés à la fois ?

- ◯ A : WindowCreate (0x15);
- ◯ B : WindowCreate (MAXIMISABLE && MINIMISABLE && RESIZABLE && ICONISABLE);
- ◯ C : WindowCreate (MAXIMISABLE | MINIMISABLE | RESIZABLE | ICONISABLE);
- ◯ D : WindowCreate ((MAXIMISABLE | MINIMISABLE) & (RESIZABLE | ICONISABLE));
- ◯ E : WindowCreate (ICONISABLE);

5 - Une petite dernière  : un site utilise un système de droits tel que décrit dans le tuto. Avant de laisser un utilisateur accéder à une page clé, je veux m'assurer qu'il ait à la fois les droits d'administrateur (gestion des membres) et de webmaster (gestion de la base).

Code : PHP

```
// Les droits du membre connecté sont dans la variable $lDroits
        // [...]
        define ('VALID_NEW',           0x10);
        define ('GER_MEMBRES',   0x20);    // administrateur
        define ('GER_BASE',            0x40);    // webmaster
        // [...]

        // Vérification des droits de l'utilisateur connecté
        if ( $Mon_operation ) {
                // Accès refusé
                exit ('Désolé, mais je suis le seul à pouvoir
accéder à cette page !');
        }
```

Par quoi dois-je remplacer $Mon_operation pour cela ?

- ◯ A : $Mon_operation = (int)$lDroits & (GER_MEMBRES | GER_BASE) ;
- ◯ B : $Mon_operation = ! ((int)$lDroits & (GER_MEMBRES | GER_BASE)) ;
- ◯ C : $Mon_operation = ! (int)$lDroits | (GER_MEMBRES | GER_BASE) ;
- ◯ D : $Mon_operation = ((int)$lDroits & (GER_MEMBRES | GER_BASE)) ^ (GER_MEMBRES | GER_BASE) ;
- ◯ E : Ce n'est pas possible avec des opérateurs de bits.

Correction !

Statistiques de réponses au QCM

J'espère vous avoir donné envie d'explorer le monde des opérateurs de bits.

Nous ne sommes pas obligés de ne stocker que des options de type oui / non. Nous pouvons aussi utiliser des valeurs sur plusieurs bits (4 choix utilisent 2 bits, 8 choix utilisent 3 bits, etc.). Le tout est d'utiliser les masques correspondants lors des opérations.

Vous trouverez d'autres exemples d'applications des opérateurs de bits dans ce cours de javascript, au paragraphe sur les opérateurs binaires.

Personnellement, j'ai mis assez longtemps avant de comprendre comment cela fonctionnait, mais je n'avais que des explications succinctes en anglais (ce n'était pas pour du PHP).

J'ai eu l'idée de ce « petit » tuto lorsque je me suis aperçu, en travaillant sur un intranet, que j'avais fait des erreurs dans ma gestion des droits...

Revenir au binaire lorsque l'on n'est plus trop sûr (sur un petit bout de papier) !

Ce tutoriel a été corrigé par les zCorrecteurs.