

Exemples de programació

Llista d'exemples de programació i utilitats en PHP.

Categories: Exemples de programació

Categories: Plataforma PHP

Com connectar-se a l'Oracle corporatiu via PDO

Per connectar-se amb l'Oracle corporatiu amb PHP utilitzem PDO, que és una capa de connexió que funciona per sobre dels diferents drivers de connexió de forma transparent per al programador. Això ens permet utilitzant la mateixa sintaxi de connexió independentment del driver que estiguem utilitzant per sota (OCI8, etc...) o de la base de dades a la que ens vulguem connectar (Oracle, MySQL, etc...).

A més, PDO ofereix altres funcionalitats com les sentències precompilades, accés a les dades mitjançant POO, etc. que podem fer-ne us. Podeu llegir el manual complet a: <http://php.net/manual/en/book.pdo.php> [1]

Per connectar-nos als tres entorns que tenim a la Diputació de Barcelona (SIT -desenvolupament-, TQA -preproducció-integració-), ORI -producció-) utilitzarem els TSNames definits en el servidor, per tant, no caldrà especificar ni IPs ni ports, només el nom de la base de dades. A més, cal notar, que pel disseny de ROLS del nostre sistema corporatiu cal llençar la instrucció "SET ROLE ALL" després de connectar-nos per tal que l'usuari adquireixi tots els drets necessaris.

Així el codi de connexió quedarà de la següent forma:

```
try {  
    $dbh = new PDO('oci:dbname=ori;charset=UTF8', 'usuari', 'password');  
    $dbh->exec('SET ROLE ALL');  
    $dbh->exec('ALTER SESSION SET NLS_COMP=LINGUISTIC');  
    $dbh->exec('ALTER SESSION SET NLS_SORT=CATALAN_AI');  
    $dbh->exec("ALTER SESSION SET NLS_TERRITORY='CATALONIA'");  
}  
catch (PDOException $e) {  
    echo 'Error de connexió: ' . $e->getMessage();  
};
```

per connectar-nos a la resta d'entorns només caldrà substituir el *dbname=ori* per *dbname=tqa* o *dbname=sit*.

Un cop connectats a la base de dades podrem realitzar qualsevol consulta SQL amb PDO i recuperar les dades a través dels mètodes que ens proporciona PDO, per exemple:

```
$stmt = $dbh->prepare('SELECT camp1, camp2 FROM taula1');  
$stmt->execute();  
$resultat = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

Si fem INSERTS, UPDATES, etc. també podem treballar amb transaccions:

```
$dbh->beginTransaction();  
...  
$dbh->commit();
```

Categories: Exemples de programació

Categories: Plataforma PHP

Consultes preparades amb PDO

PDO permet preparar les consultes que llençarem contra la base de dades abans de ser executades i definir els paràmetres o variables que utilitzarem a la sentència, són les anomenades "bind variables".

Això ens permet escapar correctament les variables de forma segura i alhora llençar la mateixa consulta "precompilada" varies vegades modificant només els valor de les "bind variables" sense haver de construir la consulta de nou, i aconseguint una notable millora en el rendiment respecte a la situació en que construïssim una consulta nova cada cop.

Veiem-ne un exemple:

```
// Construïm i preparem la consulta
$sql = 'SELECT * FROM regio WHERE habitants > :habitants AND tipus = :tipus';
$stmt = $dbh->prepare($sql, array(PDO::ATTR_CURSOR => PDO::CURSOR_FWDONLY));

// Recuperem els municipis de mes de 1.000 habitants
$stmt->execute(array(':habitants' => 1000, ':tipus' => 'municipi'));
$municipis = $stmt->fetchAll();

// Recuperem les comarques de mes de 100.000 habitants
$stmt->execute(array(':habitants' => 100000, ':tipus' => 'comarca'));
$comarques = $stmt->fetchAll();

// Recuperem les vegueries de mes de 500.000 habitants
$stmt->execute(array(':habitants' => 500000, ':tipus' => 'vegueria'));
$vegueries = $stmt->fetchAll();
```

Fixem-nos que només em definit un sol cop la consulta, la hem preparat, i tot seguit hem pogut anar recuperant varies vegades les dades modificant les variables que havíem definit sense haver de tornar a construir la consulta.

Categories: Exemples de programació

Categories: Plataforma PHP

Utilització de variables static per millorar el rendiment

Hi ha casos on l'ús de variables static en PHP pot ajudar a millorar enormement el rendiment d'una funció o mètode. És altament convenient utilitzar-ho, per exemple, quan tenim una funció que pot ser cridada varies vegades durant la generació d'una pàgina, i aquesta necessita fer alguna consulta a la base de dades o algun procés de càlcul relativament costós.

La utilització de variables static a dins d'una funció ens pot ajudar a evitar l'ús de variables globals sense perdre rendiment, una pràctica gens recomanada actualment perquè les variables globals oculten les dependències, dificulten el testeig i la depuració, etc.

Un exemple molt clar podria ser una funció que ens torna les dades de l'usuari, per exemple:

```
/**
 * Torna totes les dades del compte d'usuari passant-li l'identificador de l'usuari uid.
 *
 * @param $uid
 * (int) codi de l'usuari.
 */
function dades_usuari($uid) {
    $sql = 'SELECT * FROM usuaris WHERE uid = ?';
```

```
$stmt = $dbh->prepare($sql);  
$stmt->execute(array($uid));  
$usuari = $stmt->fetch();  
return $usuari;  
}
```

Si necessitem les dades d'un mateix usuari varies vegades en diferents parts del codi no cal que executem la mateixa consulta varis cops, ni cal que desem aquesta informació en una variable externa global. És una bona pràctica utilitzar la mateixa funció per recuperar les dades però implementar una memòria estàtica interna a la funció que farà que només executi la consulta el primer cop que li demanis la informació, i la resta de vegades servirà directament el resultat sense necessitat de repetir cap consulta.

```
/**  
 * Torna totes les dades del compte d'usuari passant-li l'identificador de l'usuari uid.  
 *  
 * @param $uid  
 * (int) codi de l'usuari.  
 */  
function dades_usuari($uid) {  
    static $usuari;  
    if (!isset($usuari[$uid])) {  
        $sql = 'SELECT * FROM usuaris WHERE uid = ?';  
        $stmt = $dbh->prepare($sql);  
        $stmt->execute(array($uid));  
        $usuari[$uid] = $stmt->fetch();  
    }  
    return $usuari[$uid];  
}
```

Ara fixeu-vos que amb aquest codi si cridem dos cops la mateixa funció i li demanem les dades del mateix usuari uid = 3, el primer cop executarà la consulta a la base de dades i desarà el resultat en la variable \$usuari[3], el segon cop que li demanem les mateixes dades les tornarà directament sense realitzar cap consulta, i si després li demanem les dades d'un usuari diferent uid = 2, aquest farà de nou la consulta perquè les dades d'aquest usuari encara no les havíem demanat i les guardarà a \$usuari[2].

Cal notar que el valor de les variables static es manté durant l'execució d'un mateix script o una pàgina, però que no es conserven entre execucions diferents. Per tant, si és una aplicació web, simplement apretant F5 o recarregant una pàgina estem resetejant totes els variables static. És un mètode per conservar dades durant la generació d'una pàgina, no durant tota una sessió.

Aquesta implementació ens permet cridar múltiples vegades una mateixa funció durant l'execució o generació d'una pàgina gestionant les dades de forma intel·ligent, tot i així, aquest codi té un defecte important. Imaginem-nos que en la lògica del nostre codi el que es fa es recuperar les dades d'un usuari, actualitzar aquestes dades i després tornar-les a recuperar en algun altre lloc. Utilitzant aquesta funció el que ens passaria és que el primer cop consultaria les dades i les guardaria en una variable estàtica, després actualitzaríem les dades i quan tornessin a cridar la funció ens serviria les dades desactualitzades desades a la variable estàtica. Per tant, cal que habilitem un mecanisme per netejar o resetejar-la. Això ho podem fer en qualsevol moment amb un simple:

```
static $usuari;  
// Esborra les dades estàtiques de l'usuari $uid  
unset($usuari[$uid]);  
// Esborra totes les dades estàtiques d'usuaris  
unset($usuari);
```

Per tant, serà molt adient que quan actualitzem les dades de l'usuari també resetegem les dades estàtiques d'aquell usuari. A més, també es útil habilitar alguna manera de recuperar les dades reals, obviat les dades

estàtiques quan cridem la funció de recuperació de dades. La forma més senzilla de fer-ho pot ser amb un paràmetre opcional.

```
/**
 * Torna totes les dades del compte d'usuari passant-li l'identificador de l'usuari uid.
 *
 * @param $uid
 * (int) codi de l'usuari.
 * @param $reset
 * (opcional) Si es TRUE reseteja les dades estàtiques d'aquell usuari. Per defecte FALSE.
 */
function dades_usuari($uid, $reset = FALSE) {
    static $usuari;
    if ($reset || !isset($usuari[$uid])) {
        $sql = 'SELECT * FROM usuaris WHERE uid = ?';
        $stmt = $dbh->prepare($sql);
        $stmt->execute(array($uid));
        $usuari[$uid] = $stmt->fetch();
    }
    return $usuari[$uid];
}
```

Ara podem forçar el reseteig de les dades estàtiques de l'usuari simplement passant un TRUE com a segon paràmetre, i decidir en tot moment si volem recuperar els dades normalment i forçar una recuperació de la base de dades sense memòria estàtica.

```
// Recupera les dades de l'usuari 11
$a = dades_usuari(11);
// Recupera els dades de l'usuari 11, aquest cop sense consultar la base
// de dades perquè ja ho has demanat abans
$b = dades_usuari(11);
// Recupera les dades de l'usuari 11, forçant la regeneració de la memòria estàtica
$c = dades_usuari(11, TRUE);
```

Nota: Aquests exemples són peces de codi simplifiades amb l'objectiu d'entendre l'ús de les variables estàtic, no són exemples per copiar literalment i utilitzar com a metodologia de treball. En un projecte real es molt adient crear una classe i implementar tots els mètodes necessaris per recuperar, crear o editar usuaris utilitzant variables estàtiques en els mètodes que sigui aplicable.

Categories: Plataforma PHP

Categories: Exemples de programació

URL d'origen: <https://comunitatdstsc.diba.cat/wiki/exemples-de-programacio>

Enllaços:

[1] <http://php.net/manual/en/book.pdo.php>