PHP OBJECT GENERATOR

ESSENTIALS

PHP Object Generator is an object-oriented code generator for PHP 4 and PHP 5 which uses the Object Relational Mapping (ORM) programming pattern to accelerate web development. ORM allows developers to 'forget' about the underlying database and instead think about their web application in terms of objects. Normally, implementing this programming pattern makes the application easier to maintain in the long run, but since the initial code is generated for you, POG also gives you a head start.

The PHP objects are generated along with 5 CRUD Methods. The first 4 CRUD methods allow you to Save(), SaveNew(), Get () and Delete() objects easily to and from your database. The 5th CRUD method, GetList(), allows you to retrieve a list of objects from the database that meet certain conditions.

What follows is a condensed introduction to each CRUD method as well as code samples that will show you how to use the generated CRUD methods to accelerate web development in PHP.

SAVE()

The Save() CRUD method allows you to insert an object into your database. If the object already exists in the database, the object will be updated instead.

Save() returns the object Id of the inserted/updated object.

CODE EXAMPLES:

To save an object, simply do the following:

```
$book = new Book();  //create a book object
$book -> title = "Php is cool";  //assign a title to the book
$book -> Save();  //this will insert the following record into the table
$book -> title = "Php is very cool"; //modify the title of the book
$book -> Save();  //this will update the record
```

You can also use the return value of the Save function for error handling:

```
$book = new Book();  //create a book object
$book -> title = "Php is cool";  //assign a title to the book
$book -> Save();  //this will insert the following record into the table
$book -> title = "Php is very cool"; //modify the title of the book
if  ($book -> Save())
{
   echo "book successfully saved";
}
else
{
   echo "something bad happened, couldn't save";
}
```

A common scenario is when you have to create a log in system to allow returning users to access their account.

```
$user = new User(); //create a user object
//get a list of user with username supplied
$user-> GetUserList("username", "=", $_POST["username"]);
//assuming all usernames are unique, this should give you the correct user
$currentUser = $userList[0];
if ($currentUser -> password == $_POST['password'])
{
    //log user in either by putting user id in the session or serializing the entire user
    $_SESSION['userId'] = $currentUser -> userId; // either put userid in session
    $_SESSION['user'] = serialize($currentUser); // or serialize user in session
}
else
{
   //user is not logged in
    echo "wrong username / password";
}
```

Another common scenario is when once a user is logged in, you want to allow him/her to modify some aspects of his/her profile on your website;

```
$user= new User(); //create a user object
$user -> Get($_SESSION['userId']); //Get the logged in user
//If you serialized the user in the example above, you can simply unserialize it
$user = unserialize($_SESSION['user']);
$user -> firstName = $_POST['firstName'];
$user -> lastName = $_POST['lastName'];
$user -> Save(); //this will update the user profile by updating the new firstname and lastname
supplied
```

GET()

The Get() CRUD method allows you to retrieve an object from your database, and must be supplied the Id of the object you want to retrieve. Since POG objects map to rows in a database table, you can think of GET as a method that allows you to fetch a specific row from your table, given that you specify the object id.

Get($id) returns the entire object.

<u>**CODE EXAMPLES:**</u>

To retrieve an object, simply do the following:

```
$book = new Book();  //instantiate a book object
$book->Get(1)  //Gets book whose id is 1.
echo $book->title;  //This should output "Php is very cool"
```

SAVENEW()

The SaveNew() CRUD method allows you to clone an object and save it to your database. SaveNew() can also be used in situations where you want to force an INSERT, rather than let POG decide whether to INSERT or UPDATE

SaveNew() returns the object Id of the inserted / updated object.

CODE EXAMPLES:

To clone an object, simply do the following:

```
$book = new Book(); //create a book object
$book->title = "Php is cool"; //assign a title to the book
$book->Save(); //this will insert the this object in to the table
$book->title = "Php is very cool"; //modify the title of the book
$book->SaveNew(); //this will create a 2nd record.
```

GETLIST()

The GetList() CRUD method allows you to return a list of objects from your database using specific conditions. The GetList() method supports specifying multiple conditions, sorting and limiting the result set.

GetList takes 1 required parameter and 3 optional ones (optional parameters are in brackets).

GetList(**$conditions *[,$sortBy] [,$ascending] [,$limit]***) returns an array of objects.

$conditions is an array of arrays containing the conditions you want to place on the query and should look like this:

```
    $conditions =
          array(
                  array("attribute1", "comparator1", "value1"),
                  array("attribute2", "comparator2", "value2"),
                  ...
                  )
```

Where

"**attribute**" (**case sensitive**) is the name of the attribute, for e.g.

    bookTitle

"**comparator**" can be any valid SQL comparator. For e.g.

    =, >, <, >=, <=, <>, LIKE (only if db encoding is turned off)

"**value**" is the value of the condition, for e.g.

    "Harry Potter"

GetList then generates the appropriate SQL statement containing the conditions joined by **AND**. For eg:

```
$conditions =
	array(
		array("attribute1", " =", "value1"),
		array("attribute2", " =", "value2"),
		...
		)
```

Will generate a SQL statement which looks like:

*Select * from table where attribute1 = value1* ***AND*** *attribute2 = value2.*

As from POG 2.5, Get List() can also generate SQL queries in Disjunctive Normal Form, i.e.:

*Select * from table where attribute1 = value1* ***OR*** *attribute2 = value2.*

The GetList () parameters to pass are, in this case:

```
array(
		array("attribute1", " =", "value1"),
		array("OR"),
		array("attribute2", " =", "value2"),
		...
		)
```

Note that when generating DNF statements, the order in which you pass the conditions matters. For example:

```
array(
		array("attribute1", " =", "value1"),
		array("attribute2", " =", "value2"),
		array("OR"),
		array("attribute3", " =", "value3"),
		...
		)
```

is equivalent to

*Select * from table where* ***(*** *attribute1 = value1* ***AND*** *attribute2 = value2****) OR (****attribute3 = value3****)***

whereas

```
array(
            array("attribute1", " =", "value1"),
            array("OR"),
            array("attribute2", " =", "value2"),
            array("attribute3", " =", "value3"),
            ...
            )
```

is equivalent to

*Select * from table where* ***(*** *attribute1 = value1****) OR (****attribute 2=value2* ***AND*** *attribute3 = value3****)***

CODE EXAMPLES

To get a list of users whose age >10

```
$user = new User(); //instantiate a user object
$userList = $user -> GetList(array(array("age", ">", "10")));
```

To get a list of users whose age >10 and who logged in more than 20 times, you'd write something like this:

```
$user = new User();
$userList =
$user -> GetList(
                    array(
                           array("age", ">", "10"),
                           array("logInCount", ">", 20)
```

```
                              )
                    );
```

To get a list of <u>all</u> books from your database, simply do the following:

```
$book = new Book(); //create a book object
$bookList = $book->GeList(array(array("bookid", ">", 0)));
foreach ($bookList as $book)
{
   echo $book->title;
}
```

To get the 10 most recent news objects from your database:

```
$news= new News(); //create a news object;
$newsList=
       $news->GetList(
                          array(array("newsid", ">", 0)),
                          "date", // tells POG to sort by the date column
                          false, // tells POG to order by descending order
                          "10" // tells POG to only return 10 results
                          );
foreach($newsList as $news)
{
   echo $news -> title;
}
```

To get the total number of books whose price is above $100 OR below $5:

```
$book= new Book(); //create a book object
// returns a list of book objects whose price > 100
$bookList= $book->GetList(
                                  array(
                                           array("price", ">", "100"),
                                        array(“OR”),
                                        array(“price”, “<”, “5”)
                                        )
echo count($bookList); //counts the number of object in the list
```

Using GetList for pagination:

```
$book= new Book(); //create a book object
// the following command is useful if for example, you are listing 10 books per html page.
//To get the results for page 1 you would do something like this:
$bookList= $book->GetList(
                                array(array("price", ">", "100")),
                                "", //tells POG to ignore this variable
                                "", //tells POG to ignore this variable
                                "0, 10" //tells POG to return 10 rows starting from row 0
                                );
//To get the results for page 2 you would do something like this:
$bookList= $book->GetList(
                                array(array("price", ">", "100")),
                                "", //tells POG to ignore this variable
                                "", //tells POG to ignore this variable
                                "10, 10" //tells POG to return 10 rows starting from row 0
                                );
//To get the results for page 3 you would do something like this:
$bookList= $book->GetList(
                                array(array("price", ">", "100")),
                                "", //tells POG to ignore this variable
                                "", //tells POG to ignore this variable
                                "20, 10" //tells POG to return 10 rows starting from row 0
                                );
```

DELETE()

The Delete() CRUD method allows you to delete an object from your database.

Delete() returns true if the deletion was completed successfully, false otherwise

CODE EXAMPLES:

To delete an object, simply do the following:

```
$book = new Book(); //create a book object
$book -> Get(1); //gets book whose id is 1
$book -> Delete(); //deletes the record from the database.
```

DELETELIST()

The Delete List() CRUD method allows you to delete all objects from your database that match certain conditions .

DeleteList(**$conditions**)

$conditions is an array of arrays containing the conditions you want to place on the query and should look like this:

```
$conditions =
    array(
        array("attribute1", "comparator1", "value1"),
        array("attribute2", "comparator2", "value2"),
        ...
        )
```

Where

"**attribute**" is the name of the attribute, for e.g.

```
bookTitle
```

"**comparator**" can be any valid SQL comparator. For e.g.

```
=,  >, <, >=, <=, <>, LIKE (only if db encoding is turned off)
```

"**value**" is the value of the condition, for e.g.

```
"Harry Potter"
```

DeleteList then generates the appropriate SQL statement containing the conditions joined by **AND**. For eg:

```
$conditions =
    array(
        array("attribute1", "=", "value1"),
        array("attribute2", "=", "value2"),
                ...
                )
```

Will generate a SQL statement which looks like:

delete from table where attribute1 = value1 ***AND*** *attribute2 = value2.*

As from POG 2.5, DeleteList() can also generate SQL queries in Disjunctive Normal Form, i.e.:

Delete from table where attribute1 = value1 ***OR*** *attribute2 = value2.*

The DeleteList() parameters to pass are, in this case:

```
array(
                array("attribute1", " =", "value1"),
                array("OR"),
                array("attribute2", " =", "value2"),
                ...
                )
```

Note that when generating DNF statements, the order in which you pass the conditions matters. For example:

```
array(
                array("attribute1", " =", "value1"),
                array("attribute2", " =", "value2"),
                array("OR"),
                array("attribute3", " =", "value3"),
                ...
                )
```

is equivalent to

delete from table where ***(*** *attribute1 = value1* ***AND*** *attribute2 = value2****) OR*** ***(****attribute3 = value3****)***

whereas

```
array(
            array(“attribute1”, “ =”, “value1”),
            array(“OR”),
            array(“attribute2”, “ =”, “value2”),
            array(“attribute3”, “ =”, “value3”),
            ...
            )
```

is equivalent to

delete from table where ***(*** *attribute1 = value1****) OR (****attribute 2=value2* ***AND*** *attribute3 = value3****)***

CODE EXAMPLES

To delete all users whose age >10

```
$user = new User(); //instantiate a user object
$user -> DeleteList(array(array("age", ">", "10")));
```

To delete a list of users whose age >10 <u>and</u> who logged in more than 20 times, you’d write something like this:

```
$user = new User();
$userList =
$user -> DeleteList(
            array(
                  array("age", ">", "10"),
                  array("logInCount", ">", 20)
```

To get the total number of books whose price is above $100 OR below $5:

```
$book= new Book(); //create a book object
$book->DeleteList(
                  array(
array("price", ">", "100"),
array(“OR”),
array(“price”, “<”, “5”)
)
```

USEFUL TIPS

USE SETUP

Use the POG setup script to create the database tables and perform some unit tests on your objects.A video of the setup process can be downloaded here:
http://www.phpobjectgenerator.com/plog/file_download/9

NAMING CONVENTIONS

POG lets you name your object and attributes any way you want. However, for best results, we suggest you generate all your objects using the same naming conventions. This is what we suggest

- Capitalize the first letter of the object name.
- Use camel casing or use the underscore character for your attributes. For e.g.

attributeName *or* attribute_name

MODIFYING & REGENERATING OBJECTS

Very often, as your requirements evolve, you will find the need to modify your objects by adding/removing attributes. There are 2 easy ways to achieve to regenerate your objects. In each object source code, POG generates and adds a URL in the header of the source code. It should look like this

```
@link: http://phpobjectgenerator.com/?objectName=&attributeList=&typeList=
```

Following this link in your browser should take you to the POG homepage with all the objectname and attributes pre-filled for you.You can then add, remove and even reorder attributes (using the UP and DOWN arrow keys).

You can also click on “Regenerate Table” in the setup interface, which will open a browser to the same location.

WE LOVE FEEDBACK

Don't be shy, come and talk to us on the POG Google Group:

http://groups.google.com/group/Php-Object-Generator

Subscribe to our RSS feed to obtain the latest news:

http://www.phpobjectgenerator.com/plog/rss/

Send us a hello through email:

pogguys@phpobjectgenerator.com