**How to Paginate Data With PHP**

by [Jason](https://tutsplus.com/authors/jason)6 Feb 2009

Difficulty:IntermediateLength:LongLanguages:

[PHP](https://code.tutsplus.com/categories/php)[Web Development](https://code.tutsplus.com/categories/web-development)

I can remember years ago when I first began coding in PHP and MySQL, how excited I was the first time I got information from a database to show up in a web browser.

For someone who had little database and programming knowledge, seeing those table rows show up onscreen based on the code I wrote (okay so I copied an example from a book -- let's not split hairs) gave me a triumphant high. I may not have fully understood all the magic at work back then, but that first success spurred me on to bigger and better projects.

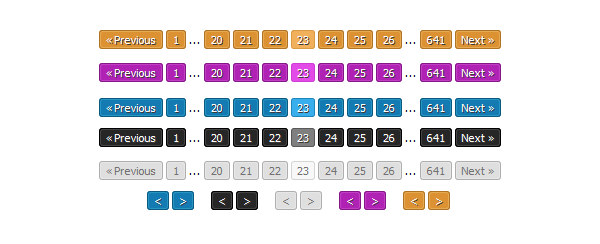
While my level of exuberance over databases may not be the same as it once was,  
ever since my first 'hello world' encounter with PHP and MySQL I've been hooked  
on the power of making things simple and easy to use.

As a developer, one problem I'm constantly faced with is taking a large set of information and making it easy to digest. Whether its a large company's client list or a personal mp3 catalog, having to sit and stare at rows upon rows upon rows of data can be discouraging  
and frustrating. What can a good developer do? Paginate!

**Looking for a Quick Solution?**

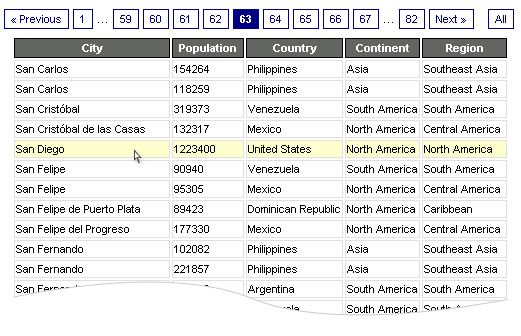
If you're looking for a quick solution, there's a great collection of [pagination scripts and helpers](https://codecanyon.net/tags/pagination) over at Envato Market.

This [PHP pagination class](https://codecanyon.net/item/php-pagination-class/72158) is really useful. It can be implemented with any PHP supported database engine, it’s very easy to customize and implement, supports two displays by default, has multicolor design schemes, and more.

[](https://codecanyon.net/item/php-pagination-class/72158)

**1. Pagination**

Pagination is essentially the process of taking a set of results and spreading  
them out over pages to make them easier to view.



I realized early on that if I had 5000 rows of information to display not only  
would it be a headache for someone to try and read, but most browsers would take  
an Internet eternity (i.e. more than about five seconds) to display it.

To solve this I would code various SQL statements to pull out chunks of data, and if I was  
in a good mood I might even throw in a couple of "next" and "previous" buttons.  
After a while, having to drop this code into every similar project and customize  
it to fit got old. Fast. And as every good developer knows, laziness breeds inventiveness  
or something like that. So one day I sat down and decided to come up with a simple,  
flexible, and easy to use PHP class that would automatically do the dirty work for  
me.

A quick word about me and PHP classes. I'm no object-oriented whiz. In fact, I hardly  
ever use the stuff. But after reading some OOP examples and tutorials, and some  
simple trial and error examples, I decided to give it a whirl and you know what?  
It works perfectly for pagination. The code used here is written in PHP 4 but will  
work in PHP 5.

**2. The Database**

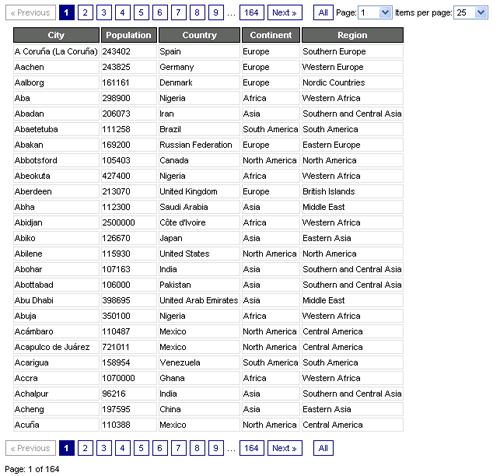
Gotta love MySQL. No offense to the other database systems out there, but for  
me, all I need is MySQL. And one great feature of MySQL is that they give you some  
free sample databases to play with at <http://dev.mysql.com/doc/#sampledb>.

For my examples, I'll be using the world database (~90k zipped) which contains over  
4000 records to play with, but the beauty of the PHP script we'll be creating is  
that it can be used with any database. Now I think we can all agree that if we decided  
not to paginate our results that we would end up with some very long and unwieldy  
results like the following:



(click for full size, ridiculously long image ~ 338k)

So lets gets down to breaking up our data into easy to digest bites like this:



Beautiful isn't it? Once you drop the pagination class into your code you can  
quickly and easily transform a huge set of data into easy to navigate pages with  
just a few lines of code. Really.

**3. Paginator**

This example will be composes of two scripts, the reusable paginator class and the index file that will display the table items and controls.

**Paginator.class.php**

The paginator class will have only two methods and the constructor, we will build it gradually explaining each step as we move forward.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11 | <?php    class Paginator {         private $\_conn;          private $\_limit;          private $\_page;          private $\_query;          private $\_total;    } |

This definition only set's the paginator required member variables, since this is a helper class and it's destined for pagination only it will rely on a valid connection to the MySQL server and an already defined query that we will append the parameters necessary to paginate the results. We'll start with the constructor method.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11 | <?php    public function \_\_construct( $conn, $query ) {        $this->\_conn = $conn;      $this->\_query = $query;        $rs= $this->\_conn->query( $this->\_query );      $this->\_total = $rs->num\_rows;    } |

Quite simple right? This method only set's the object's database connection and the necessary query, after that it calculates the total number of rows retrieved by that query without any limit nor skip parameters, this total is necessary to create the links for the paginator.

Note that for simplicity we are not doing error checking or any other validation of the given parameters, but in a real world application this checks will be necessary.

**Retrieving Results**

Now let's create the method that will actually paginate the data and return the results.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | <?php  public function getData( $limit = 10, $page = 1 ) {        $this->\_limit   = $limit;      $this->\_page    = $page;        if ( $this->\_limit == 'all' ) {          $query      = $this->\_query;      } else {          $query      = $this->\_query . " LIMIT " . ( ( $this->\_page - 1 ) \* $this->\_limit ) . ", $this->\_limit";      }      $rs             = $this->\_conn->query( $query );        while ( $row = $rs->fetch\_assoc() ) {          $results[]  = $row;      }        $result         = new stdClass();      $result->page   = $this->\_page;      $result->limit  = $this->\_limit;      $result->total  = $this->\_total;      $result->data   = $results;        return $result;  } |

Let's analyze this one step at a time, first we set the limit and page parameters, which by default are set the 10 and 1 respectively. The we check if the user is requiring a given number of rows or all of them, base on this and the page parameter we set the LIMIT parameter of the query, the "- 1" of the page is taking into account the fact that we are starting the pages in 1 instead of 0.

After this we simply evaluate the query and get the results, finally we create a new results object which contain the limit, page and total parameters of the executed query as well as the data for each of the retrieved rows.

**Displaying Pagination Links**

Now let's write the method used to get the pagination links.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | <?php  public function createLinks( $links, $list\_class ) {      if ( $this->\_limit == 'all' ) {          return '';      }        $last       = ceil( $this->\_total / $this->\_limit );        $start      = ( ( $this->\_page - $links ) > 0 ) ? $this->\_page - $links : 1;      $end        = ( ( $this->\_page + $links ) < $last ) ? $this->\_page + $links : $last;        $html       = '<ul class="' . $list\_class . '">';        $class      = ( $this->\_page == 1 ) ? "disabled" : "";      $html       .= '<li class="' . $class . '"><a href="?limit=' . $this->\_limit . '&page=' . ( $this->\_page - 1 ) . '">&laquo;</a></li>';        if ( $start > 1 ) {          $html   .= '<li><a href="?limit=' . $this->\_limit . '&page=1">1</a></li>';          $html   .= '<li class="disabled"><span>...</span></li>';      }        for ( $i = $start ; $i <= $end; $i++ ) {          $class  = ( $this->\_page == $i ) ? "active" : "";          $html   .= '<li class="' . $class . '"><a href="?limit=' . $this->\_limit . '&page=' . $i . '">' . $i . '</a></li>';      }        if ( $end < $last ) {          $html   .= '<li class="disabled"><span>...</span></li>';          $html   .= '<li><a href="?limit=' . $this->\_limit . '&page=' . $last . '">' . $last . '</a></li>';      }        $class      = ( $this->\_page == $last ) ? "disabled" : "";      $html       .= '<li class="' . $class . '"><a href="?limit=' . $this->\_limit . '&page=' . ( $this->\_page + 1 ) . '">&raquo;</a></li>';        $html       .= '</ul>';        return $html;  } |

This is a rather long method, something like 34 lines of code, so now let's explain what is going on in this method.

1. First we evaluate if the user is requiring a given number of links or all of them, in the second case we simply return an empty string since no pagination is required.
2. After this we calculate the last page based on the total number of rows available and the items required per page.
3. Then we take the links parameter which represents the number of links to display below and above the current page, and calculate the start and end link to create.
4. Now we create the opening tag for the list and set the class of it with the list class parameter and add the "previous page" link, note that for this link we check if the current page is the first, and if so, we set the disabled property of the link.
5. At this point we display a link to the first page and an ellipsis symbol in case that the start link is not the first one.
6. Next we add the links below and above the current page based on the previously calculated start and end parameters, in each step we evaluate the current page again the link page displayed and set the active class accordingly.
7. After this we display another ellipsis symbol and the link to the last page in case that the end link is not the last one.
8. Finally we display the "next page" link and set the disabled state when the user is viewing the last page, close the list and return the generated HTML string.

That's all there is to the Paginator.class, of course we could add setters and getters for the database connection, limit, page, query and total parameters but for simplicity we'll keep it this way.

**4. Index.php**

Now we'll create the file in charge of using the Paginator class and displaying the data, so first let me show you the base HTML.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | <!DOCTYPE html>      <head>          <title>PHP Pagination</title>          <link rel="stylesheet" href="css/bootstrap.min.css">      </head>      <body>          <div class="container">                  <div class="col-md-10 col-md-offset-1">                  <h1>PHP Pagination</h1>                  <table class="table table-striped table-condensed table-bordered table-rounded">                          <thead>                                  <tr>                                  <th>City</th>                                  <th width="20%">Country</th>                                  <th width="20%">Continent</th>                                  <th width="25%">Region</th>                          </tr>                          </thead>                          <tbody></tbody>                  </table>                  </div>          </div>          </body>  </html> |

Quite simple, this file only displays a table that we will populate with the information retrieved from the database, note that for this example I'm using bootstrap for basic page styling.

**Using The Paginator**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?php for( $i = 0; $i < count( $results->data ); $i++ ) : ?>          <tr>                  <td><?php echo $results->data[$i]['Name']; ?></td>                  <td><?php echo $results->data[$i]['Country']; ?></td>                  <td><?php echo $results->data[$i]['Continent']; ?></td>                  <td><?php echo $results->data[$i]['Region']; ?></td>          </tr>  <?php endfor; ?> |

Now for make use of our Paginator class add the following php code at the top of the document.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14 | <?php      require\_once 'Paginator.class.php';        $conn       = new mysqli( '127.0.0.1', 'root', 'root', 'world' );        $limit      = ( isset( $\_GET['limit'] ) ) ? $\_GET['limit'] : 25;      $page       = ( isset( $\_GET['page'] ) ) ? $\_GET['page'] : 1;      $links      = ( isset( $\_GET['links'] ) ) ? $\_GET['links'] : 7;      $query      = "SELECT City.Name, City.CountryCode, Country.Code, Country.Name AS Country, Country.Continent, Country.Region FROM City, Country WHERE City.CountryCode = Country.Code";        $Paginator  = new Paginator( $conn, $query );        $results    = $Paginator->getData( $page, $limit );  ?> |

This script is quite simple, we just required or Paginator class, note that this code assumes that this file is in the same directory as the index.php file, if this is not the case you should update the path accordingly.

The we create the connection to our database using the MySQLi library, retrieve the paginator parameters from the GET request and set the query, since this is not an article on MySQL or any ot that I will not get into details about the connection or the query used here.

Lastly we create the Paginator object and retrieve the results for the current page.

**Displaying The Results**

Now to display the obtained results add the following code to the table body.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?php for( $i = 0; $i < count( $results->data ); $i++ ) : ?>          <tr>                  <td><?php echo $results->data[$i]['Name']; ?></td>                  <td><?php echo $results->data[$i]['Country']; ?></td>                  <td><?php echo $results->data[$i]['Continent']; ?></td>                  <td><?php echo $results->data[$i]['Region']; ?></td>          </tr>  <?php endfor; ?> |

Here we simple are iterating through the results data attribute containing the cities records and creating a table row for each one of them.

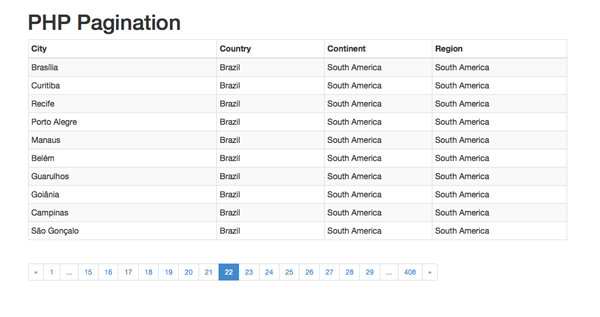
Advertisement

**Pagination Links**

Now to display the paginator links add the following code below the table.

|  |  |
| --- | --- |
| 1 | <?php echo $Paginator->createLinks( $links, 'pagination pagination-sm' ); ?> |

To the Paginator createLinks method we pass the obtained *links* parameter and the css class for the pagination links used from bootstrap. Here is the result of the created page.



**Conclusion**

This should provide you with everything that you need to know in order to get up and running with pagination in your application.

Please don't hesitate to leave questions, comments, or general feedback in the feed below!

Still stuck? Try contacting one of our [experienced PHP developers](https://studio.envato.com/search?utf8=%E2%9C%93&search%5Bquery%5D=php) on Envato Studio.