Joomla provides a sophisticated database abstraction layer to simplify the usage for third party developers. This guide will help you use this layer.

Why should I use the Joomla database class?

Joomla can use different kinds of SQL database systems and run in a variety of environments with different table-prefixes. In addition to these functions, the class automatically creates the database connection. Besides instantiating the object you need just two lines of code to get a result from the database in a variety of formats. Using the Joomla database layer ensures a maximum of compatibility and flexibility for your extension.

Preparing the query

```
// Get a database object
$db =& JFactory::getDBO();

$query = "SELECT * FROM #__example_table WHERE id = 999999;";
$db->setQuery($query);
```

First we instantiate the database object; then we prepare the query. You can use the normal SQL syntax. The only thing you have to change is the table prefix. To make this as flexible as possible, Joomla uses a placeholder for the prefix, the “#_”. In the next step, the $db→setQuery(), this string is replaced with the correct prefix.

Now, if we don't want to get information from the database, but rather insert a row into it, we need one more function. Every string value in the SQL syntax should be quoted. For example, MySQL uses backticks `` for names and single quotes ‘‘ for values. Joomla has some functions to do this for us and to ensure code compatibility between different databases. We can pass the names to the function $db→nameQuote($name) and the values to the function $db→Quote($value).

A fully quoted query example is:

```
$query = "
  SELECT *
    FROM ".$db->nameQuote('#__example_table')."
    WHERE ".$db->nameQuote('id')." = ".$db->quote('999999').";
  ";
```

Whatever we want to do, we have to set the query with the $db→setQuery() function. Although you could write the query directly as a parameter for $db→setQuery(), it's commonly done by first saving it in a variable, normally $query, and then handing this variable over. This results in clean, readable code.

setQuery($query)

The `setQuery($query)` method sets up a database query for later execution either by the query() method or one of the Load result methods.

```
$db =& JFactory::getDBO();
$query = "/* some valid sql string */";
$db->setQuery($query);
```

Notes: The parameter $query must be a valid SQL string. It can either be added as a string parameter or as a variable. Generally a variable is preferred; it leads to more legible code and can help in debugging.

setQuery() also takes three other parameters: $offset, $limit (both used in list pagination) and $prefix, an alternative table prefix. All three variables have default values set and can usually be ignored.

Executing the Query

To execute the query, Joomla provides several functions, which differ in their return value.

Basic Query Execution

query()

The `query()` method is the the basic tool for executing SQL queries on a database. In Joomla it is most often used for updating or administering the database simply because the various load methods detailed on this page have the query step built into them.

The syntax is very straightforward:

```
$db =& JFactory::getDBO();
$query = "/* some valid sql string */";
$db->setQuery($query);
$result = $db->query();
```

Note: $query() returns an appropriate database resource if successful, or FALSE if not.

Query Execution Information

* getAffectedRows() * explain() * insertid()

Insert Query Execution

* insertObject()

Query Results

The database class contains many methods for working with a query's result set.

1 style="background:yellow"	John Smith	johnsmith@example.com	
2	Magda Hellman	magda_h@example.com	
3	Yvonne de Gaulle	ydg@example.com	

Single Value Result

loadResult()

This is often the result of a 'count' query to get a number of records:
Use '`loadResult()`' when you expect just a single value back from your database query.

```
$db =& JFactory::getDBO();
$query = "
  SELECT COUNT(*)
    FROM ".$db->nameQuote('#__my_table')."
    WHERE ".$db->nameQuote('name')." = ".$db->quote($value).";
  ";
$db->setQuery($query);
$count = $db->loadResult();
```

{| class="wikitable" style="text-align:center"

or where you are just looking for a single field from a single row of the table (or possibly a single field from the first row returned).

```
$db =& JFactory::getDBO();
$query = "
  SELECT ".$db->nameQuote('field_name')."
    FROM ".$db->nameQuote('#__my_table')."
    WHERE ".$db->nameQuote('some_name')." = ".$db->quote($some_value).";
  ";
$db->setQuery($query);
$result = $db->loadResult();
<PHP>

===Single Row Results ===
Each of these results functions will return a single record from the
database even though there may be several records that meet the criteria
that you have set. To get more records you need to call the function again.
{| class="wikitable" style="text-align:center"
|-
! id !! name !! email !! username
|- style="background:yellow"
| 1 || John Smith || johnsmith@example.com || johnsmith
|-
| 2 || Magda Hellman || magda_h@example.com || magdah
|-
| 3 || Yvonne de Gaulle || ydg@example.com || ydegaulle
|}

==== loadRow() ====
loadRow() returns an indexed array from a single record in the table:
<PHP>
. . .
$db->setQuery($query);
$row = $db->loadRow();
print_r($row);
```

will give: <pre>Array ([0] ⇒ 1 [1] ⇒ John Smith [2] ⇒ johnsmith@example.com [3] ⇒ johnsmith) </pre>

You can access the individual values by using:<pre>$row['index'] *e.g. $row['2']</pre> Notes: # The array indices are numeric starting from zero. # Whilst you can repeat the call to get further rows, one of the functions that returns multiple rows might be more useful. ==== loadAssoc() ==== loadAssoc() returns an associated array from a single record in the table: <PHP> . . . $db→setQuery($query); $row = $db→loadAssoc(); print_r($row); </PHP> will give: <pre>Array ([id] ⇒ 1 [name] ⇒ John Smith [email] ⇒ johnsmith@example.com [username] ⇒ johnsmith)</pre> You can access the individual values by using:<pre>$row['name']* e.g. $row['name']</pre>

Notes: # Whilst you can repeat the call to get further rows, one of the functions that returns multiple rows might be more useful.

loadObject()

loadObject returns a PHP object from a single record in the table:

```
. . .
$db->setQuery($query);
$result = $db->loadObject();
print_r($result);
```

will give: <pre>stdClass Object ([id] ⇒ 1 [name] ⇒ John Smith [email] ⇒ johnsmith@example.com [username] ⇒ johnsmith)</pre>

You can access the individual values by using:<pre>$row→index *e.g. $row→email</pre> Notes: # Whilst you can repeat the call to get further rows, one of the functions that returns multiple rows might be more useful. ===Single Column Results === Each of these results functions will return a single column from the database. {| class="wikitable" style="text-align:center" |- ! id !! name !! email !! username |- | 1 || style="background:yellow" | John Smith || johnsmith@example.com || johnsmith |- | 2 || style="background:yellow" | Magda Hellman || magda_h@example.com || magdah |- | 3 || style="background:yellow" | Yvonne de Gaulle || ydg@example.com || ydegaulle |} ==== loadResultArray() ==== loadResultArray() returns an indexed array from a single column in the table: <PHP> $query = " SELECT name, email, username FROM . . . "; . . . $db→setQuery($query); $column= $db→loadResultArray(); print_r($column); </PHP> will give: <pre>Array ([0] ⇒ John Smith [1] ⇒ Magda Hellman [2] ⇒ Yvonne de Gaulle)</pre> You can access the individual values by using:<pre>$column['index']* e.g. $column['2']</pre>

Notes: # The array indices are numeric starting from zero. # loadResultArray() is equivalent to loadResultArray(0).

loadResultArray($index)

loadResultArray($index) returns an indexed array from a single column in the table:

```
$query = "
  SELECT name, email, username
    FROM . . . ";
```

```
. . .
$db->setQuery($query);
$column= $db->loadResultArray(1);
print_r($column);
```

will give: <pre>Array ([0] ⇒ johnsmith@example.com [1] ⇒ magda_h@example.com [2] ⇒ ydg@example.com)</pre>

You can access the individual values by using:<pre>$column['index'] *e.g. $column['2']</pre> loadResultArray($index) allows you to iterate through a series of columns in the results <PHP> . . . $db→setQuery($query); for ($i = 0; $i ⇐ 2; $i++) { $column= $db→loadResultArray($i); print_r($column); } </PHP> will give: <pre>Array ([0] ⇒ John Smith [1] ⇒ Magda Hellman [2] ⇒ Yvonne de Gaulle) Array ([0] ⇒ johnsmith@example.com [1] ⇒ magda_h@example.com [2] ⇒ ydg@example.com) Array ([0] ⇒ johnsmith [1] ⇒ magdah [2] ⇒ ydegaulle)</pre> </pre> Notes: # The array indices are numeric starting from zero. === Multi-Row Results === Each of these results functions will return multiple records from the database. {| class="wikitable" style="text-align:center" |- ! id !! name !! email !! username |- style="background:yellow" | 1 || John Smith || johnsmith@example.com || johnsmith |- style="background:yellow" | 2 || Magda Hellman || magda_h@example.com || magdah |- style="background:yellow" | 3 || Yvonne de Gaulle || ydg@example.com || ydegaulle |} ==== loadRowList() ==== loadRowList() returns an indexed array of indexed arrays from the table records returned by the query: <PHP> . . . $db→setQuery($query); $row = $db→loadRowList(); print_r($row); </PHP> will give (with line breaks added for clarity): <pre>Array ([0] ⇒ Array ([0] ⇒ 1 [1] ⇒ John Smith [2] ⇒ johnsmith@example.com [3] ⇒ johnsmith) [1] ⇒ Array ([0] ⇒ 2 [1] ⇒ Magda Hellman [2] ⇒ magda_h@example.com [3] ⇒ magdah) [2] ⇒ Array ([0] ⇒ 3 [1] ⇒ Yvonne de Gaulle [2] ⇒ ydg@example.com [3] ⇒ ydegaulle))</pre> You can access the individual rows by using:<pre>$row['index']* e.g. $row['2']</pre> and you can access the individual values by using:<pre>$row['index']['index'] *e.g. $row['2']['3']</pre> Notes: # The array indices are numeric starting from zero. ==== loadAssocList() ==== loadAssocList() returns an indexed array of associated arrays from the table records returned by the query: <PHP> . . . $db→setQuery($query); $row = $db→loadAssocList(); print_r($row); </PHP> will give (with line breaks added for clarity): <pre>Array ([0] ⇒ Array ([id] ⇒ 1 [name] ⇒ John Smith [email] ⇒ johnsmith@example.com [username] ⇒ johnsmith) [1] ⇒ Array ([id] ⇒ 2 [name] ⇒ Magda Hellman [email] ⇒ magda_h@example.com [username] ⇒ magdah) [2] ⇒ Array ([id] ⇒ 3 [name] ⇒ Yvonne de Gaulle [email] ⇒ ydg@example.com [username] ⇒ ydegaulle)) </pre> You can access the individual rows by using:<pre>$row['index']* e.g. $row['2']</pre> and you can access the individual values by using:<pre>$row['index']['column_name'] *e.g. $row['2']['email']</pre> ==== loadAssocList($key) ==== loadAssocList('key') returns an associated array - indexed on 'key' - of associated arrays from the table records returned by the query: <PHP> . . . $db→setQuery($query); $row = $db→loadAssocList('username'); print_r($row); </PHP> will give (with line breaks added for clarity): <pre>Array ([johnsmith] ⇒ Array ([id] ⇒ 1 [name] ⇒ John Smith [email] ⇒ johnsmith@example.com [username] ⇒ johnsmith) [magdah] ⇒ Array ([id] ⇒ 2 [name] ⇒ Magda Hellman [email] ⇒ magda_h@example.com [username] ⇒ magdah) [ydegaulle] ⇒ Array ([id] ⇒ 3 [name] ⇒ Yvonne de Gaulle [email] ⇒ ydg@example.com [username] ⇒ ydegaulle))</pre> You can access the individual rows by using:<pre>$row['key_value']* e.g. $row['johnsmith']</pre> and you can access the individual values by using:<pre>$row['key_value']['column_name'] *e.g. $row['johnsmith']['email']</pre> Note: Key must be a valid column name from the table; it does not have to be an Index or a Primary Key. But if it does not have a unique value you may not be able to retrieve results reliably. ==== loadObjectList() ==== loadObjectList() returns an indexed array of PHP objects from the table records returned by the query: <PHP> . . . $db→setQuery($query); $result = $db→loadObjectList(); print_r($result); </PHP> will give (with line breaks added for clarity): <pre>Array ([0] ⇒ stdClass Object ([id] ⇒ 1 [name] ⇒ John Smith [email] ⇒ johnsmith@example.com

[username] ⇒ johnsmith) [1] ⇒ stdClass Object ([id] ⇒ 2 [name] ⇒ Magda Hellman [email] ⇒ magda_h@example.com [username] ⇒ magdah) [2] ⇒ stdClass Object ([id] ⇒ 3 [name] ⇒ Yvonne de Gaulle [email] ⇒ ydg@example.com [username] ⇒ ydegaulle))</pre> You can access the individual rows by using:<pre>$row['index'] e.g. $row['2']</pre> and you can access the individual values by using:<pre>$row['index']→name *e.g. $row['2']→email</pre> ==== loadObjectList('key') ==== loadObjectList($key) returns an associated array - indexed on 'key' - of objects from the table records returned by the query: <PHP> . . . $db→setQuery($query); $row = $db→loadObjectList('username'); print_r($row); </PHP> will give (with line breaks added for clarity): <pre>Array ([johnsmith] ⇒ stdClass Object ([id] ⇒ 1 [name] ⇒ John Smith [email] ⇒ johnsmith@example.com [username] ⇒ johnsmith) [magdah] ⇒ stdClass Object ([id] ⇒ 2 [name] ⇒ Magda Hellman [email] ⇒ magda_h@example.com [username] ⇒ magdah) [ydegaulle] ⇒ stdClass Object ([id] ⇒ 3 [name] ⇒ Yvonne de Gaulle [email] ⇒ ydg@example.com [username] ⇒ ydegaulle)) </pre> You can access the individual rows by using:<pre>$row['key_value']* e.g. $row['johnsmith']</pre> and you can access the individual values by using:<pre>$row['key_value']→column_name *e.g. $row['johnsmith']→email</pre> Note: Key must be a valid column name from the table; it does not have to be an Index or a Primary Key. But if it does not have a unique value you may not be able to retrieve results reliably. === Miscellaneous Result Set Methods === ==== getNumRows() ==== getNumRows() will return the number of result rows found by the last query and waiting to be read. To get a result from getNumRows() you have to run it '`after`' the query and '`before`' you have retrieved any results. <PHP> . . . $db→setQuery($query); $db→query(); $num_rows = $db→getNumRows(); print_r($num_rows); $result = $db→loadRowList(); </PHP> will return <pre>3</pre> Note: if you run getNumRows() after loadRowList() - or any other retrieval method - you may get a PHP Warning: <pre>Warning: mysql_num_rows(): 80 is not a valid MySQL result resource in D:\xampp\htdocs\joomla1.5a\libraries\joomla\database\database\mysql.php on line 344</pre> ==Tips, Tricks & FAQ== ===Subqueries=== We had a few people lately using subqueries like these: <PHP> SELECT * FROM #*example WHERE id IN (SELECT id FROM #example2); </PHP> These kinds of queries are only possible in MySQL 4.1 and above. Another way to achieve this, is splitting the query into two: <PHP> $query = "SELECT id FROM #example2"; $database→setQuery($query); $query = "SELECT * FROM #example WHERE id IN (". implode(",", $database→loadResultArray()) .")"; </PHP> ===Developer-Friendly Tips=== Here is a quick way to do four developer-friendly things at once: * Use a simple constant as an SQL seperator (which can probably be used in many queries). * Make your SQL-in-PHP code easy to read (for yourself and possibly other developers later on). * Give an error inside your (component-) content without really setting debugging on. * Have a visibly nice SQL by splitting SQL groups with linebreaks in your error. <PHP> $db =& JFactory::getDBO(); $jAp=& JFactory::getApplication();* We define a linebreak constant define('L', chr(10));

```
  //Here is the most magic
```

$db→setQuery(

```
'SELECT * FROM #__table'.L.
'WHERE something="something else")'.L.
'ORDER BY date desc'
```

); $db→query();

```
  //display and convert to HTML when SQL error
```

if (is_null($posts=$db→loadRowList())) {$jAp→enqueueMessage(nl2br($db→getErrorMsg()),'error'); return;} </PHP> DevelopmentDatabase

From:
https://wiki.merkatu.info/ - **Wiki de merkatu**

Permanent link:
https://wiki.merkatu.info/jdatabase?rev=1285591149

Last update: **2017/03/27 17:43**