Database

Overview

HUBzero has been built with the ability to use several different kinds of SQL-database-systems and to 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, at a minimum, you only need 2 lines of code to get a result from the database in a variety of formats. Using the database layer ensures a maximum of compatibility and flexibility for your extension.

This tutorial looks at how to set and execute various queries.

Configuration

The database configuration for a hub is located at app/config/database.php.

Connections

A hub establishes a database connection, by default, with the configuration specified in app/config/database.php but alternate connections may be established.

```
$mydb = HubzeroDatabaseDriver::getInstance([
    'driver'   => 'pdo',
    'host'     => 'example',
    'user'     => 'example',
    'password' => '******',
    'database' => 'mystuff',
    'prefix'   => 'hub_'
]);
```

Preparing The Query

```
// Get a database object
$db = App::get('db');

$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 insert a row into it, we need one more function. Every string-value in the SQL-syntax should be quoted. For example, MySQL uses back-ticks `` 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 $db->quoteName($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 helps writing 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 = App::get('db');
$query = "/* some valid sql string */";
$db->setQuery($query);
```

Note: 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 as 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 of these 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

The query() method is the the basic tool for executing sql queries on a database. In the CMS it is most often used for updating or administering the database and not seen often for loading data. This largely because the various load methods detailed on this page have the query step built in to them.

The syntax is very straightforward:

```
$db = App::get('db');
$query = "/* some valid sql string */";
$db->setQuery($query);
$result = $db->query();
```

Note: $db->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.

Single Value Result

loadResult()

Use loadResult() when you expect just a single value back from your database query.

id	name	email	username
1	John Smith	johnsmith@example.com	johnsmith
2	Magda Hellman	magda_h@example.com	magdah
3	Yvonne de Gaulle	ydg@example.com	ydegaulle

This is often the result of a 'count' query to get a number of records:

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

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 = App::get('db');
$query = "
  SELECT ".$db->nameQuote('field_name')."
    FROM ".$db->nameQuote('#__my_table')."
    WHERE ".$db->nameQuote('some_name')." = ".$db->quote($some_va
lue).";
  ";
$db->setQuery($query);
$result = $db->loadResult();
```

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.

id	name	email	username
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:

```
...
$db->setQuery($query);
$row = $db->loadRow();
print_r($row);
```

will give:

```
Array (
 [0] => 1
 [1] => John Smith
 [2] => johnsmith@example.com
 [3] => johnsmith
)
```

You can access the individual values by using:

```
$row['index'] // e.g. $row['2']
```

Note:

1. The array indices are numeric starting from zero.
2. 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:

```
$db->setQuery($query);
$row = $db->loadAssoc();
print_r($row);
```

will give:

```
Array (
 [id] => 1
 [name] => John Smith
 [email] => johnsmith@example.com
 [username] => johnsmith
)
```

You can access the individual values by using:

```
$row['name'] // e.g. $row['name']
```

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:

```
stdClass Object (
 [id] => 1
 [name] => John Smith
 [email] => johnsmith@example.com
 [username] => johnsmith
)
```

You can access the individual values by using:

```
$row->index // e.g. $row->email
```

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.

id	name	email	username
1	John Smith	johnsmith@example.com	johnsmith
2	Magda Hellman	magda_h@example.com	magdah
3	Yvonne de Gaulle	ydg@example.com	ydegaulle

loadColumn()

loadColumn() returns an indexed array from a single column in the table:

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

$db->setQuery($query);
$column= $db->loadColumn();
print_r($column);
```

will give:

```
Array (
 [0] => John Smith
 [1] => Magda Hellman
 [2] => Yvonne de Gaulle
)
```

You can access the individual values by using:

```
$column['index'] // e.g. $column['2']
```

Note:

1. The array indices are numeric starting from zero.
2. loadColumn() is equivalent to loadcolumn(0)

loadColumn($index)

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

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

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

will give:

```
Array (
 [0] => johnsmith@example.com
 [1] => magda_h@example.com
 [2] => ydg@example.com
)
```

You can access the individual values by using:

```
$column['index'] // e.g. $column['2']
```

loadColumn($index) allows you to iterate through a series of columns in the results

```
$db->setQuery($query);
for ( $i = 0; $i <= 2; $i++ ) {
  $column= $db->loadColumn($i);
  print_r($column);
}
```

will give:

```
Array ( [0] => John Smith [1] => Magda Hellman [2] => Yvonne de G
aulle )
Array ( [0] => johnsmith@example.com [1] => magda_h@example.com [
2] => ydg@example.com )
Array ( [0] => johnsmith [1] => magdah [2] => ydegaulle )
```

The array indices are numeric starting from zero.

Multi-Row Results

Each of these results functions will return multiple records from the database.

id	name	email	username

id	name	email	username
1	John Smith	johnsmith@example.com	johnsmith
2	Magda Hellman	magda_h@example.com	magdah
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:

```
$db->setQuery($query);
$row = $db->loadRowList();
print_r($row);
```

will give:

```
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 )
)
```

You can access the individual values by using:

```
$row['index'] // e.g. $row['2']
```

and you can access the individual values by using:

```
$row['index']['index'] // e.g. $row['2']['3']
```

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:

```
$db->setQuery($query);
$row = $db->loadAssocList();
print_r($row);
```

will give:

```
Array (
 [0] => Array ( [id] => 1 [name] => John Smith [email] => johnsmi
th@example.com [username] => johnsmith )
 [1] => Array ( [id] => 2 [name] => Magda Hellman [email] => magd
a_h@example.com [username] => magdah )
 [2] => Array ( [id] => 3 [name] => Yvonne de Gaulle [email] => y
dg@example.com [username] => ydegaulle )
)
```

You can access the individual rows by using:

```
$row['index'] // e.g. $row['2']
```

and you can access the individual values by using:

```
$row['index']['column_name'] // e.g. $row['2']['email']
```

loadAssocList($key)

loadAssocList($key) returns an associated array - indexed on 'key' - of associated arrays from the table records returned by the query:

```
$db->setQuery($query);
$row = $db->loadAssocList('username');
print_r($row);
```

will give:

```
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 [ema
il] => ydg@example.com [username] => ydegaulle )
)
```

You can access the individual rows by using:

```
$row['key_value'] // e.g. $row['johnsmith']
```

and you can access the individual values by using:

```
$row['key_value']['column_name'] // e.g. $row['johnsmith']['email
']
```

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:

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

will give:

```
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 )
)
```

You can access the individual rows by using:

```
$row['index'] // e.g. $row['2']
```

and you can access the individual values by using:

```
$row['index']->name // e.g. $row['2']->email
```

loadObjectList('key')

loadObjectList('key') returns an associated array - indexed on 'key' - of objects from the table records returned by the query:

```
$db->setQuery($query);
$row = $db->loadObjectList('username');
print_r($row);
```

will give:

```
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 G
aulle
     [email] => ydg@example.com [username] => ydegaulle )
)
```

You can access the individual rows by using:

```
$row['key_value'] // e.g. $row['johnsmith']
```

and you can access the individual values by using:

```
$row['key_value']->column_name // e.g. $row['johnsmith']->email
```

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.

Misc 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.

```
$db->setQuery($query);
$db->query();
$num_rows = $db->getNumRows();
```

```
print_r($num_rows);
$result = $db->loadRowList();
```

will give:

3

Note: if you run getNumRows() after loadRowList() - or any other retrieval method - you may get a PHP Warning.