DBQUERY.ORG

dbQuery

Rutger Plak (rutger.plak@dbquery.org)

Documentation

Author:
Rutger PLAK

Company:
Anan6

August 16, 2012

Documentation

In this document, the functionality of the clientside javascript software of dbQuery is documented. For every function, the parameters, return values and functionality is defined. Each of the functions is assisted with examples.

How it works

dbQuery is a Javascript framework that allows you access to any `MySql` database by using plain Javascript. You don't have to know any serverside programming language to access the data stored in your `MySQL` database. You can access any table by simply calling the `db()` function.

The `db()` function will return a javascript object (or collection of) with its attributes typed and initialized as defined in the database. The returned object will also have the methodes `getAttribute()`, `setAttribute()`, `save()` and `remove()`, which will allow you to add, modify and delete any row stored in the database.

To initialize a `db()` object you'll have to pass 3 parameters to the `db()` function:

- **Selector** The database and table name of the table you want to access.
- **Filter** A filter object allowing you to retreive certain rows according to your filter criteria. By default the primary key is used as a filter.
- **Callback** This function will be called after the rows are retreived from the database to javascript. The callback function receives a single parameter: an array of javascript row objects.

Installation

You need 6 files to use dbQuery:

Serverside

- **dbquery.config** Global configuration file.
- **dbObject.inc.php** Class dbObject, used by dbQuery's PHP serverside software.
- **dbAttribute.inc.php** Class dbAttribute, used by dbQuery's PHP serverside software.
- **q.php** Webservice that can be accessed by the world. This is the only file on the server that has to be accessible.

Clientside

- **jquery.js** dbQuery uses jQuery for making calls to the server.
- **dbquery.js** Handles all dbQuery functionality.

The files `dbObject.inc.php`, `dbAttribute.inc.php`, `dbquery.config` and `q.php` have to be placed on the server. `q.php` has to be accessible. In the HTML pages where dbQuery is used, `jquery.js` and `dbquery.js` have to be included in the header using the next code:

```
<head>
  <script type="text/javascript" src="/path/to/jquery.js"></script>
  <script type="text/javascript" src="/path/to/dbquery.js"></script>
</head>
```

Configuration

In the file `dbquery.config`, security details have to be declared in order for dbQuery to be allowed to access the database. There are several variables:

Required

- `$__HOST_OF_THE_DATABASE`
  The hostname of the database. Can be either an IP-adress or a tracable URL. Default: `localhost`.
- `$__USER_OF_THE_DATABASE`
  Username that dbQuery uses to communicatie with the database.
- `$__PASSWORD_OF_THE_DATABASE`
  Password for dbQuery's user for database communication.

Optional

- `$__PATH_TO_DBOBJECT`
  When `dbObject.inc.php` and `dbAttribute.inc.php` are not in the same directory as `dbquery.config`, you have to change this path.
- `$__DEFAULT_DATABASE`
  In order to use dbQuery with as most ease as possible, you can select a default database.
- `$__READ_ONLY_MODE`
  When dbQuery should only read from the database and no write access is allowed, this value has to be set to 'true'. Default is false.

Creating objects

db()

The function texttdb is the main function of dbQuery. `db` is used to create objects from the database. `db` has several parameters:

Function **db(tablename [, selection] [, handler(eventObject))**

Returns an array with objects from the database that follow the selectioncriteria or an object if the selection consists of only an integer.

- **tablename** Name of the table which has to be turned into an object. Optionally prefixed by the database name seperated with a dot.
- **selectie** The selection has two possible values:
  1. **Integer** An integer that is the identifier of the row in the database that has to be turned into an object.
  2. **Array** An array with selectioncriteria. The index of the array is the attribute (column in the database). The value at the index is the value for the selection.
- **handler(eventObject)** Callback that is executed when the data is fetched from the database.

When only a single object has to be fetched from the database, `db` can be used as constructor. When using the constructor, **eventObject** is the single object instead of the array of objects.

Example 1

In this simple example, a customer is fetched from the database and the object is logged to the console. Because it is only one customer that is fetched from the database, the constructor is used, so the eventObject consists of the customer only.

```
new db('customers',1, function(customer) {
     console.log(customer);
});
```

Example 2

In this simple example, a customer is fetched from the database and the object is logged to the console. Because the constructor is not used, an array is returned.

```
db('customers',1, function(customers) {
     console.log(customers[0]);
});
```

Example 3

In this example, all customers who are 23 years old are fetched from the database. The size of the set is logged.

```
db('customers', { 'age': 23 }, function(customers) {
     console.log(customers.length);
});
```

Example 4

In this example, all active customers of 20 years old are fetched from the database and their nickname is changed.

```
db('customers', { 'age': 20, 'status': 'active' }, function(customers) {
     customers.forEach(function(customer, i) {
        customer.setAttribute('nickname','20 year old customer number '+i).
            save();
     });
});
```

Using objects

The following functions can be used on objects fetched from the database using `db`.

dbobject.getAttribute()

The function `getAttribute` can be used to fetch an attribute of an object. When this object is a foreign key, the object where this foreign key is pointed is returned.

Function .getAttribute(**attribute**, **handler(eventObject)**)

Returns the `dbObject` where the function is called to, so the function can be chained.

- **attribute** Requested attribute (columnname).
- **handler(eventObject)** A function that is executed when the call is finished, where *eventObject* is the requested object.

Example 5

In the next example, a customer is fetched from the database. His name is logged to the console. After that, his address (which is a foreign key) is fetched, and the zipcode is logged.

```
new db('customers',1, function(customer) {
     // normall attribute request
     console.log(customer.getAttribute('name'));
     // address links to another table
     customer.getAttribute('address', function(address) {
       console.log(address.getAttribute('zipcode'));
     });
});
```

Example 6

In the next example, a callback is defined and used in `getAttribute`.

```
var remove = function(address) {
  address.remove();
}
db('customers',{'age':'23'}, function(customers) {
     customers.forEach(function(customer) {
  // Use the predefined callback
```

```
        customer.getAttribute('address', remove(address));
    });
});
```

dbobject.setAttribute()

The function setAttribute can be used to set the value of an attribute of the object.

Function .setAttribute(**attribute**, **newValue**)

Returns the `dbObject` where the function is called to, so the function can be chained.

- **attribute** Attribute (column) that has to be changed.
- **newValue** New value of the attribute (column).

Example 7

In this example, a customer is fetched from the database and his name and institute are changed.

```
new db('customers',1, function(customer) {
    // make changes to the attributes
    customer.setAttribute('name','Plak')
    .setAttribute('firstname','Rutger')
    .setAttribute('institude','University of Technology Delft');
});
```

dbobject.save()

The function `save` can be used to save changes made to the object into the database.

Function .save(**handler(eventObject)**)

Returns the `dbObject` where the function is called to, so the function can be chained. Because a `save`-call often requires post-functionality, a callback can be defined and will be executed when the object is saved into the database.

Example 8

In this example, a customer is fetched from the database. His name and institution are changed and the changes are saved to the database.

```
new db('customers',1, function(customer) {
    // making changes to the attributes
    customer.setAttribute('name','Plak')
    .setAttribute('firstname','Rutger')
    .setAttribute('institute','University of Technology Delft')
    // save
    .save();
});
```

Example 9

In this example, a new customer is created. His name and institute are changed and saved into the database, so a new row is inserted in the database. After that, the identifier of the table customers is logged to the console. This value is only available after the object is saved to the database, so the logging happens in a callback.

```
new db('customers',0, function(customer) {
     // making changes to the attributes
     customer.setAttribute('name','Plak')
     .setAttribute('firstname','Rutger')
     .setAttribute('institute','University of Technology Delft')
     // save
     .save(function(customer) {
       console.log(customer.getAttribute('customer'));
     });
});
```

dbobject.remove()

The function `remove` can be used to remove objects from the database.

Function .remove(**handler(eventObject)**)

Because a `remove`-call often requires post-functionality, a callback can be defined and will be executed when the object is deleted from the database. The `eventObject` is the old identifier of the object, as the object itself doesn't exist anymore.

Example 10

In the next example, a customer is removed from the database.

```
new db('customers',1, function(customer) {
     // delete customer with identifier 1 from the database
     customer.remove();
});
```

Example 11

In the next example, a customer is removed from the database when the delete button in the table is clicked. After it has been removed from the database, the customer is removed from the table.

```
$('#myTable tr td.delete').bind('click', function() {
  // create dbObject with the identifier of the tr of the table
  new db('customers',$(this).parent().attr('id'), function(customer) {
    // Remove the customer from the database
    customer.remove(function(customerId) {
      // remove the table row with the id of the deleted customer
      $('tr#'+customerId).remove():
    });
  });
});
```