

Table Class

Use this class for database manipulation (CRUD).

Example table Employee in database.

| Column Name | Datatype | PK | NN | UQ | BIN | UN | ZF | AI | G | Default |
|------------------|--------------|-------------------------------------|-------------------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------------------------------------|--------------------------|---------|
| id | INT(11) | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | |
| birth_date | DATETIME | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |
| first_name | VARCHAR(45) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |
| middle_name | VARCHAR(45) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |
| last_name | VARCHAR(45) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |
| gender | INT(1) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |
| hire_date | DATETIME | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |
| department | INT(11) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |
| fired | TINYINT(1) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | '0' |
| employeephone | VARCHAR(45) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |
| email | VARCHAR(100) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |
| socialsecurityno | VARCHAR(45) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |
| createUser | VARCHAR(45) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |
| createDateTime | DATETIME | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |
| updateUser | VARCHAR(45) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |
| updateDateTime | DATETIME | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |

Public functions

Create variable of class Table ([\\$employee](#)).

- void init()** - initiates class fields with default values set in database.

```
$employee->init();
```

- int insert(bool <default true>)**

```
$employee → init();
$employee → set_First_name('John');
$employee → set_Last_name('Smith');
$employee → insert();
```

Inserts row in database table with values set in class fields.
Returns inserted row id or false (0) if failure.

if you need to insert many data in several tables

```
$employee → init();
$employee → set_First_name('John');
$employee → set_Last_name('Smith');
$employee → insert(false);
$employee → init();
```

```

$employee → set_First_name('Mary');
$employee → set_Last_name('Poppins');
$employee → insert(false);
// after commit transaction or rollback if you have errors.

```

It reduces database writes.

3. void deleteById(int \$id, bool \$commit <default true>);

Removes row from database table with selected id.

```
$employee->deleteById(1);
```

4. void getById(int \$id);

initiates variable fields with values from database table by selected id.

```
$employee->getById(1);
```

5. void update(bool \$commit);

Updates row in database.

```

$employee->getById(1);
$employee->set_First_name('Poppins');
$employee->update();

```

6. void setFilter(string column, string type, [mixed firstvalue, [mixed secondvalue]])

Sets filter condition on table before select from database

```

$employee → setFilter('first_name','=', 'John');
$employee->setFilter('department','IN',array(1,2,3,4));
$employee → setFilter('hire_date','BETWEEN', $date1, $date2);
var_dump($employee->fetchAll());

```

result: All employees with first name John, working in department with ids [1,2,3,4] , hired between dates \$date1 and \$date2.

type variable possible values:

1. '=' - values in column EQUALS firstvalue
2. '>' - values in column Bigger firstvalue
3. '>=' bigger or equal
4. '<' less
5. '<=' less or equal
6. '<>' NOT Equal
7. 'BETWEEN'
8. 'NOT BETWEEN'
9. 'IN' (firstvalue may be an array or comma separated string with values)
10. 'NOT IN' (firstvalue may be an array or comma separated string with values)
11. 'LIKE' contains any part of firstvalue
12. 'LIKE _%' starts with firstvalue
13. 'LIKE %_' ends with firstvalue

7. void reset()

Resets all filters, order, limit and cursor columns.

```
$employee → reset();
```

8. Void setOrder(string columns, string ordertypes<default ASC>)

Sort table by columns in order. Ordertypes ASC or DESC

```
$employee → setOrder('first_name','DESC');
```

multiple sorting

```
$employee → setOrder('first_name,last_name', 'ASC,DESC');
```

9. void setCursorColumns(array columns);

Specify only columns you need.

```
$employee->setCursorColumns(array('first_name','last_name'));  
var_dump($employee → fetchAll());
```

result: associative array with columns id, first_name, last_name

Attention: result always contains column id

10. void setLimit(array limit)

Sets limit on records selection

```
$employee->setLimit(array(10,30));
```

result: 30 records starting from 10th

11. void getFirstRow();

Initiates variable fields with values of first row

```
$employee → setFilter('first_name','=','John');  
$employee → setOrder('hire_date','DESC');  
$employee → getFirstRow();
```

Employee fields will be initiated by values of record with name John last hired.

12. int getCount();

Returns rows count in table.

```
$employee → setFilter('first_name','=','John');  
echo $employee → getCount();
```

result: Quantity of rows with name John.

13. String `getGroupConcat(string column<default 'id'>)`

Returns commaseparated string with unique column values.

| Id | first_name | last_name |
|----|------------|-----------|
| 1 | John | Smith |
| 2 | Mary | Poppins |

```
echo $employee → getGroupConcat('first_name');  
result: 'John,Mary'
```

```
echo $employee → getGroupConcat();  
result: '1,2'
```

14. array `fetchAll();`

Returns associative array with table records.

```
$employee → setCursorColumns(array('first_name','last_name'));  
$employee → setFilter('id','IN','1,2');  
var_dump($employee → fetchAll());
```

```
result: array(0=> array('id'=>'1','first_name'=>'John','last_name'=>'Smith'),  
1=>array('id'=>'2','first_name'=>'Mary','lastname'=>'Poppins'));
```

15. array `fetchHashMap(string column<default id>)`

Returns associative array with key values from column

```
$employee → setCursorColumns(array('first_name','last_name'));  
$employee → setFilter('id','IN','1,2');  
var_dump($employee → fetchHashMap());
```

```
result: array(1=> array('id'=>'1','first_name'=>'John','last_name'=>'Smith'),  
2=>array('id'=>'2','first_name'=>'Mary','last_name'=>'Poppins'));
```

```
var_dump($employee → fetchHashMap('first_name'));
```

```
result: array('John'=> array('id'=>'1','first_name'=>'John','last_name'=>'Smith'),  
'Mary'=>array('id'=>'2','first_name'=>'Mary','last_name'=>'Poppins'));
```

Warning: if column values are not unique, result keys will be filled with last record by this key.

16. mixed `getSum(string column), getAvg(string column), getMax(string column), getMin(string column)`

Function to count values.

```
$employee → setFilter('first_name', '=', 'John');
```

```
echo $employee → getMax('hire_date');
```

result: max hire date for employees with name John.

17. void bulkInsert(array records, bool commit<default true>)

Inserts rows from records.

```
$records = array(0=> array('first_name'=> 'John','last_name'=>'Smith'),  
                1=>array('first_name'=>'Mary','last_name'=>'Poppins'));
```

```
$employee → bulkInsert($records);
```

Warning: if array keys don't match table fields these keys and values will be ignored.

18. void setSelectionView(string viewName)

Function sets view that has to be used instead table.

If you are familiar with MySQL, you may create view based on table **employees**

```
CREATE  
VIEW `employee_view` AS  
    SELECT  
        `employee`.`id` AS `id`,  
        `employee`.`birth_date` AS `birth_date`,  
        `employee`.`first_name` AS `first_name`,  
        `employee`.`middle_name` AS `middle_name`,  
        `employee`.`last_name` AS `last_name`,  
        `employee`.`gender` AS `gender`,  
        `employee`.`hire_date` AS `hire_date`,  
        `employee`.`department` AS `department`,  
        `departments`.`dept_name` AS `department_name`,  
        `employee`.`fired` AS `fired`,  
        `employee`.`employeephone` AS `employeephone`,  
        `employee`.`email` AS `email`,  
        `employee`.`socialsecurityno` AS `socialsecurityno`,  
        `employee`.`createUser` AS `createUser`,  
        `employee`.`createDateTime` AS `createDateTime`,  
        `employee`.`updateUser` AS `updateUser`,  
        `employee`.`updateDateTime` AS `updateDateTime`  
    FROM  
    (`employee`  
    LEFT JOIN `departments` ON ((`employee`.`department` = `departments`.`id`)))
```

| id | birth_date | first_name | middle_name | last_name | gender | hire_date | department | department_name | fired | employeephone |
|----|---------------------|------------|-------------|-----------|--------|---------------------|------------|-----------------|-------|----------------|
| 1 | 1930-09-05 00:00:00 | Neil | Alden | Armstrong | 1 | 1955-09-05 00:00:00 | 1 | Accounting | 1 | (895)564-97-56 |
| 2 | 1982-01-02 00:00:00 | Andrew | S | Fuller | 1 | 2016-11-04 00:00:00 | 2 | Administration | 1 | (786)546-56-46 |
| 4 | 2016-07-04 00:00:00 | John | | Smith | 1 | 2016-11-03 00:00:00 | 1 | Accounting | 1 | (879)787-87-98 |
| 5 | 2016-11-23 00:00:00 | Nancy | | Nodier | 2 | 2016-11-08 00:00:00 | 4 | IT | 1 | (786)546-56-46 |
| 7 | 2016-11-02 00:00:00 | Mike | | Lemberg | 1 | 2016-11-09 00:00:00 | 3 | Finance | 1 | (785)489-55-49 |
| 8 | 1980-02-12 00:00:00 | Alice | | York | 2 | 1986-11-01 00:00:00 | 9 | Warehouse | 1 | (899)456-46-86 |

| id | dept_name |
|----|----------------|
| 1 | Accounting |
| 2 | Administration |
| 3 | Finance |
| 4 | IT |
| 5 | Security |
| 6 | Transport |
| 7 | Logistics |

```
$employee->setSelectionView('employee_view');
$employee->setFilter('department_name','=';'Accounting');
var_dump($employee->fetchAll());
```

result: associative array containing all fields from **employee_view**

Useful if you need aggregated data connected to table, which you need sort or filter by.

```
$employee → setSelectionView(null); // all selections will be from table employee again
```

*Warning: View must contain field **id** (unique).*

***insert, update, delete** and other changing data functions still affect on physical table **employee** in database.*

*There is no setter for field **department_name**,
but you may use getter (**\$employee->get_Department_name()**).*

Setters and Getters

1. Setters and getters are generated automatically from database.

Setter: **set_<Field name>(<Value>);**

Getter: **get_<Field name>();**

attention: after «set_» or «get_» field name has to start from uppercase letter

get_First_name — correct

get_first_name - incorrect

```
$employee → getById(1);
```

```
echo $employee → get_First_name();
```

```
echo $employee → get_Last_name();
```

result: John
Smith

```
$employee → getById(1);
```

```
$employee → set_First_name('James');
```

```
$employee → update();
```

result: Changed first_name from John to James in record with id == 1

variable *\$employee* stores previous field values.

```
$employee → getById(1);  
$employee → set_First_name('James');  
$employee → update();  
echo $employee → getOld_First_name();
```

result: John (despite value has been changed in database we can get previous value of record.)

Use `getOld_<Field_name>` getters to check whether value was changed or not.

2. Instead setters you may use function `copyFieldValuesFrom()` :

(especially if you don't know how many fields you need to fill)

```
$empl = array('first_name'=>'John', 'last_name'=>'Smith');  
// associative array (with keys equal table fields) or another object of class Table
```

```
$employee → init();  
$employee → copyFieldValuesFrom($empl);  
$employee → insert();
```

Warning:

1. `copyFieldValuesFrom` doesn't copy field *id*. If you need copy all fields including *id* use function `copyAllFieldValuesFrom()`

2. keys in variable `$empl`, which don't match field names in `$employee`, will be ignored.

3. Getting table objects.

If table in database has foreign keys to another table, you may get referenced table object.

`obj_<Field_name>` (uppercase first letter).

In our case table `employee` has foreign key to table `departments`

we can get table `departments` with initiated values with `id` from field `department`

```
$department = $employee->obj_Department();
```

Warning: Table has to be initiated with values, otherwise methods return NULL

or if we need department name from specified employee

```
$employee->getById(1);  
echo $employee → obj_Department()->get_Dept_name();  
you may chain objects  
$employee_absences->obj_Employeeid()->obj_Department()->get_Dept_name();
```

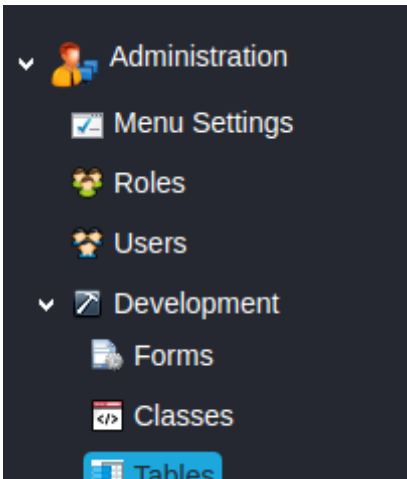
Table triggers.

Class Table has functions triggered on affecting database rows.

Functions: **onInsert()**, **onUpdate()**, **onDelete()**, **onAfterInsert()**, **onAfterUpdate()**, **onAfterDelete()**.

These functions are empty, but called every time according to actions with records. If you need to do some actions automatically triggered you may overwrite these functions in the table you need.

Select in Main Menu: Administration → Development → Tables . Select the table you need to extend. In our case it is the table Employee



Click the button «Edit Source Code».

Create class Employee. Override the functions you need. Programming language is PHP.

Every time a record in the table employee is updated, the field `updateUser` is set with the current user ID, `updateDateTime` is set with the current server datetime. Every new record, the fields `createUser` and `createDateTime` are set with corresponding values.

```
Edit Source Code employee
Save
1 <?php
2 .....class Employee extends Table{
3 .....function __construct($db_link, $tableFactory=null){
4 .....parent::__construct('employee', $db_link, $tableFactory);|
5 .....}
6 .....
7 .....protected function onUpdate(){
8 .....    $this->set_UpdateUser( security::userId() );
9 .....    $this->set_UpdateDateTime(date('Y-m-d H:i:s', time()));
10 .....}
11 .....
12 .....protected function onInsert(){
13 .....    $this->set_CreateUser(security::userId());
14 .....    $this->set_CreateDateTime(date('Y-m-d H:i:s', time()));
15 .....}
16 .....}
17 ?>
```

You may add here special functions for any table class or override existing functions.

Also class Table has helpful functions to track changes.

Bool isChanged_<Field_name>() (Field_name uppercase first letter);

protected function onUpdate(){


```
    if($this->isChanged_First_name()){  
        do something ...  
    }  
}
```

Table `getTable(string tablename)`

Creates new object of type Table.

```
protected function onUpdate(){  
    if($this->isChanged_First_name()){  
        $department = $this->getTable('departments');  
        do something...  
    }  
}
```

Forms

All forms have standart view.

The screenshot shows a data management interface with a grid of records and an actions panel. Red numbers 1-8 point to various UI elements:

- 1: Tab «View»
- 2: Grid header (No., Birth Date, First Name, Last Name, Department, Ge..., Empl. Date, Phone, Fired)
- 3: Grid row 1 (Neil Armstrong)
- 4: Grid row 2 (Andrew Fuller)
- 5: Grid row 3 (John Smith)
- 6: Grid row 4 (Nancy Nodier)
- 7: Grid row 5 (Mike Lemberg)
- 8: Grid row 6 (Alice York)

The actions panel on the right contains the following buttons: Print, Messages, TestReport, TestGrid, Save as csv, Save as, Print all, test1, Save State, load State, readfile, Encrypt.

At the bottom, there is a pagination control: Go to page: 1 Show rows: 25 1-24 of 24

1. Tab «View» contains grid with table records. Here you can browse, select, filter, sort, add, delete, edit records.

Tab «Details» contains form view of only one selected record where you can edit values of this record.

The screenshot shows the 'Details' tab with the following form fields:

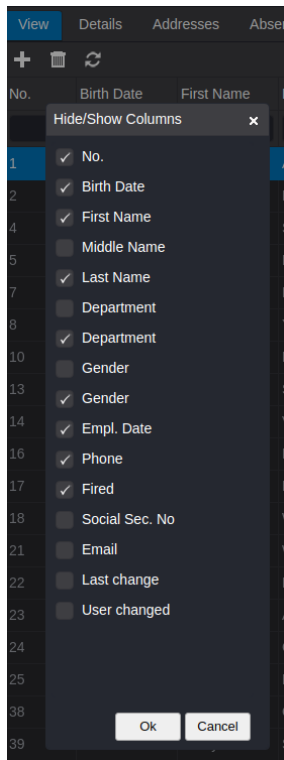
- No.: 1
- Birth Date: 08/15/2018
- First Name: Neil
- Middle Name: Alden
- Last Name: Armstrong
- Department: 11 (Production)
- Gender: M
- Empl. Date: 08/09/2018
- Phone: (895)564-97-56
- Fired:
- Social Sec. No: 123123123123
- Email: example@email.com
- Last change: 08/11/2018 13:04:55
- User changed: 1

2. Toolbar with buttons (Add, Delete, Reload)

- 3. Toolbar with column names
- 4. Filters toolbar
- 5. Tabs with connected subforms. You may place unlimited number of subforms.

| From Date | To Date | Cause of Ab... | Quantity | Unit of |
|------------|------------|----------------|----------|---------|
| 03/13/2017 | 03/16/2017 | DAYOFF | 3 | Day |
| 03/18/2017 | 03/24/2017 | HOLIDAY | 6 | Day |
| 04/26/2017 | 04/27/2017 | SICK | 1 | Day |
| 04/27/2017 | 04/27/2017 | DAYOFF | 8 | Hour |

- 6. Hide/Show button.

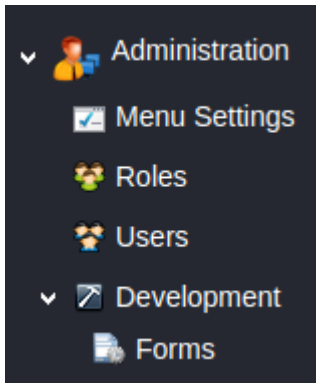


- 7. Button set. Here you may place various action buttons.

- 8. Grid paging settings.

Creating forms

We have table Employee in database. Create form Employee for this table.



Open Menu → Administration → Development → Forms

Add new record in opened form.

In field **Name** write the name of form (*employee*)

Press button «Edit Source Code»

Programming language **Javascript**.

Requirements for form.

Maindata object:

```
{
  dataSourceTableName: ``,
  fields:<array of fields objects>
}
```

fields properties

name : field name in database table **required**

type: `number`, `string`, `date`, `bool` **required**

label: the name of field displayed on a form **required**

width: the width of field **required**

editable: **true**, **false** (default **false**) **optional**

fieldformat: **required for fields of type date**

gridColumnProperties: **optional** (here you may customise grid columns. See manual JQWidgets Grid Columns)

input: **optional** (here you can customise controls displayed on tab Details).

See manual for

type: number — [jqxNumberInput](#)

type: string - [jqxInput](#)

type: date - [jqxDateTimeInput](#)

type: bool - [jqxCheckBox](#)

Example minimal code to run form:

```
var dataSourceTableName = 'employee'; // the name of table in database
```

```
// array of fields, describe every field as object
```

```
var fields = [
```

```
  { name: 'id', type: 'number', label: 'No.',width: 70},
```

```
  { name: 'birth_date', type: 'date',label: 'Birth Date', width:100 ,
```

```
  fieldformat:'MM/dd/yyyy', editable: true, gridColumnProperties:{filtertype:'range'}},
```

```
  { name: 'first_name', type: 'string',label: 'First Name', width: 100, editable:true},
```

```
  { name: 'middle_name', type: 'string',label: 'Middle Name', width: 100, editable:true, hidden:true},
```

```
  { name: 'last_name', type: 'string', label: 'Last Name', width:100, editable:true},
```

```
  { name: 'fired', type:'bool', label:'Fired', width:70, editable:true, gridColumnProperties:{columnntype:'checkbox'}}];
```

```
var maindata = {dataSourceTableName: dataSourceTableName, fields:fields};
```

```
runForm(maindata);
```

showlabel: wether show control label on tab Details (optional default TRUE)
 display: wether show control on tab Details (optional default TRUE)

dropdowngrid:

// used to fill field value with values from another table

```
{ name: 'gender', type: 'number',label: 'Gender', width:50, editable:true, gridColumnProperties:{hidden:true},
showlabel:false, display:false },

{ name: 'gender__shortname', type: 'string',label: 'Gender', width:40,

  dropdowngrid:{datatable:'gender', needfield:'id', setfield:'gender',
    fields:[ {name:'id', type:'number', label:'No.', width:70, gridColumnProperties:{hidden:true}},
      {name:'name', type:'string', label:'Sex', width:150}
    ],
    gridProperties:{showfilterrow: false, pageable: false,showheader: false}
  }
}
```

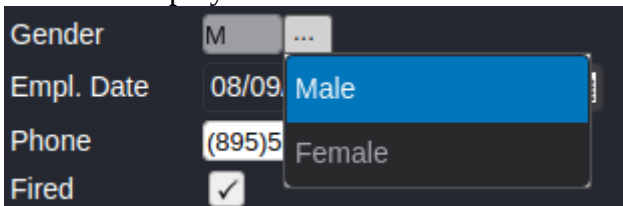
datatable: source table for selection

needfield: specify field from datatable you need

setfield: specify field from maindata to set with value from needfield (optional, default field where dropdowngrid is declared)

gridProperties: see manual for jqxGrid (jqwidgets.com)

in Form Employee



in database

| # | id | name | shortname |
|---|----|--------|-----------|
| 1 | 1 | Male | M |
| 2 | 2 | Female | F |

if table has Foreign key to another table (in our case table *employee* has foreign key to table *gender*) You can display values from referenced table using double underline in property *name*.

| Foreign Key Name | Referenced Table |
|------------------|-----------------------|
| emp_dep | `myerp`.`departments` |
| emp_gend | `myerp`.`gender` |

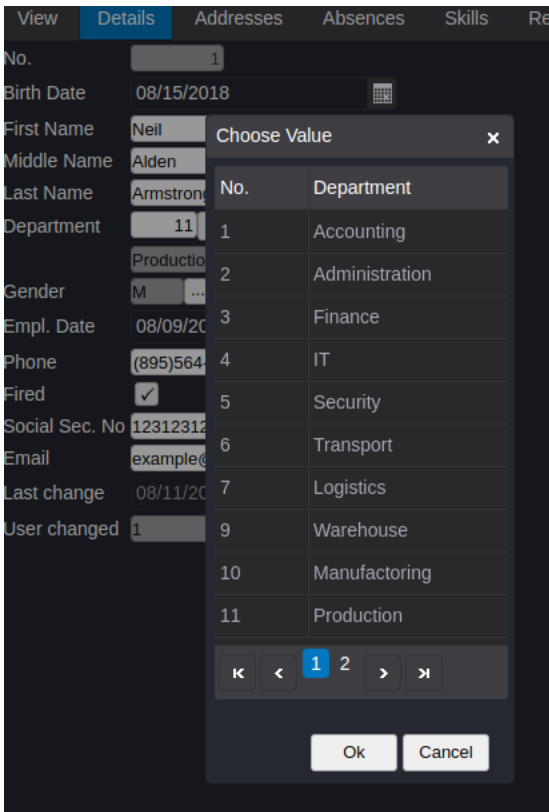
| Foreign Key Columns | |
|--|-------------------|
| Column | Referenced Column |
| <input type="checkbox"/> id | |
| <input type="checkbox"/> birth_date | |
| <input type="checkbox"/> first_name | |
| <input type="checkbox"/> middle_name | |
| <input type="checkbox"/> last_name | |
| <input checked="" type="checkbox"/> gender | id |

In database record gender field value is 1, but in form employee displayed value is M (shortcode field from table *gender*). Because we specified in object field property *name* as *gender__shortname* (<field from table *employee*><double underline><field from referenced table>)

| # | id | birth_date | first_name | middle_name | last_name | gender | hire_date | department | fired | employeephone | email |
|---|----|---------------------|------------|-------------|-----------|--------|---------------------|------------|-------|----------------|---------|
| 1 | 1 | 2018-08-15 00:00:00 | Neil | Alden | Armstrong | 1 | 2018-08-09 00:00:00 | 11 | 1 | (895)564-97-56 | example |

filterBefore: array of filters. For instance, we want to show in dropdowngrid only records with *id* greater than 1. Specify property filterBefore

filterBefore:[['id','>','1']] ([[filter1],[filter2],...,[filterN]])
 popupgrid: similar to dropdowngrid, opened in popup modal window.



Popupgrid can be a function, called with current row. If you want to open different windows depending on values of current row. For example in Purchase Quotes Lines you need to open table Service, Item, Resource, Fixed Asset depending on chosen type in purchase order line.

Example code with object:

```
{ name: 'department', type: 'string',label: 'Department', width:40,
  popupgrid:{ datatable:'department', needfield:'id',
    fields:[ {name:'id', type:'number', label:'No.', width:70},
      {name:'dept_name', type:'string', label:'Department', width:150}
    ],
    gridProperties:{showfilterrow: false, pageable: true,showheader: true}
  }
}
```

Example code with function:

in table resources we fill field employeeid only if type is «employee» (type=1)

```
{ name: 'type', type: 'number', label: 'Resource Type',width: 70,
  { name: 'employeeid', type: 'number', label: 'Employee',width: 70,
    popupgrid:function(currentrow){
      if(currentrow.type!=1)return undefined; //no popup window if type!=1
      return {datatable:'employee', needfield:'id',
        fields:[
          {name:'id', type:'number', label:'No.', width:70},
          {name:'first_name', type:'string', label:'First Name', width:150},
          {name:'last_name', type:'string', label:'Last Name', width:150}
        ]
      };
    }
  }
}
```

gridProperties: [see manual for jqxGrid](#)

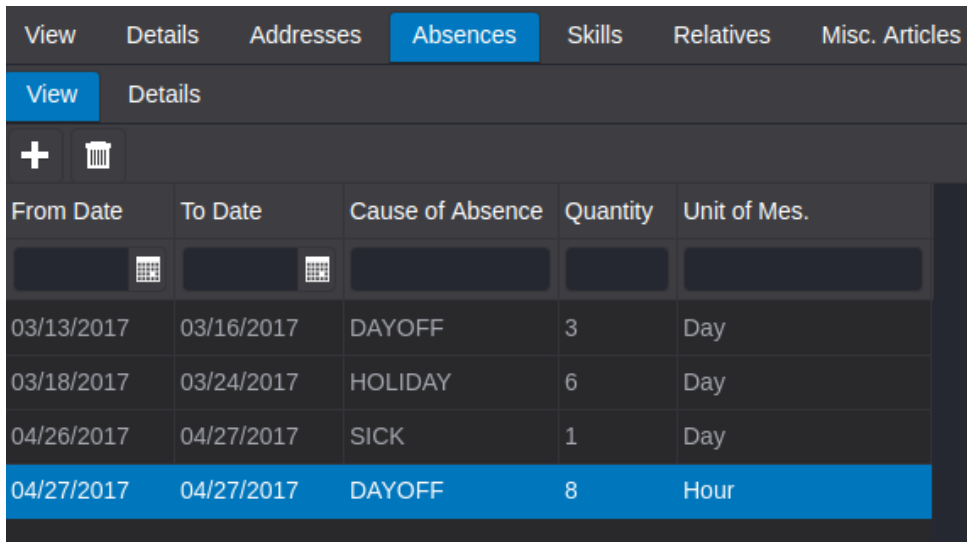
Subforms.

```
// array of objects
```

```
var subdata = [sub1, sub2... subN];
```

subdata object:

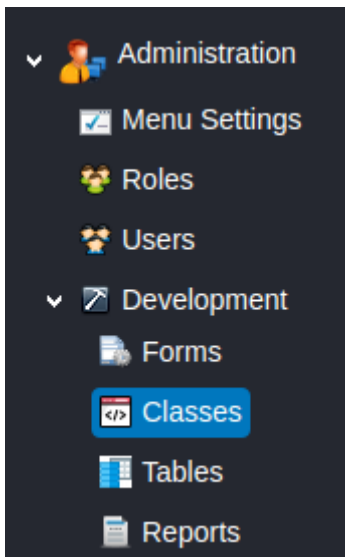
```
{ name: 'absences', label: 'Absences', label: 'No.',width: 70},
```



The screenshot shows a web application interface with a dark theme. At the top, there is a navigation bar with tabs: 'View', 'Details', 'Addresses', 'Absences' (selected), 'Skills', 'Relatives', and 'Misc. Articles'. Below this, there is a sub-navigation bar with 'View' (selected) and 'Details'. The main content area contains a table with the following columns: 'From Date', 'To Date', 'Cause of Absence', 'Quantity', and 'Unit of Mes.'. The table has four rows of data, with the last row highlighted in blue. The first two columns have calendar icons for date selection.

| From Date | To Date | Cause of Absence | Quantity | Unit of Mes. |
|------------|------------|------------------|----------|--------------|
| 03/13/2017 | 03/16/2017 | DAYOFF | 3 | Day |
| 03/18/2017 | 03/24/2017 | HOLIDAY | 6 | Day |
| 04/26/2017 | 04/27/2017 | SICK | 1 | Day |
| 04/27/2017 | 04/27/2017 | DAYOFF | 8 | Hour |

Classes



Administration → Development → Classes

In opened form add new record. Write class name you want to add.
Click Button **Edit Source Code**

Programming language is PHP.

```
Edit Source Code noseriesmanagement
Save
1 <?php
2
3 class NoSeriesManagement extends baseclass{
4     function getNextNo($series_code,$commit=false){
5         ..... $no_series = $this->getTable('no_series');
6         ..... $no_series->setFilter('code','=', $series_code);
7         ..... if(!$no_series->getFirstRow())return null;
8
9         ..... $last_no = null;
10        ..... $next_no = null;
11        ..... if(is_null($no_series->get_Last_no_used())||($no_series->get_Last_no_used()=='')){
12            ..... $last_no=$no_series->get_Starting_no();
13            ..... $next_no=$last_no;
14        } else{
15            ..... $last_no = $no_series->get_Last_no_used();
16            ..... $next_no = $this->incstr($last_no,$no_series->get_Increment(),$no_series->get_Ending_no());
17        }
18
19        ..... $no_series->set_Last_no_used($next_no);
20        ..... $no_series->update($commit);
21
22        ..... return $next_no;
23    }
24
25 }
```

If your class will interact with database, extend it from **baseclass**.

Now you may use functions:

[getTable\(<tablename>\)](#) - creates object of class Table

[getClass\(<classname>\)](#) - creates object of another Class

[beginTransaction\(\)](#) - begins transaction

[commitTransaction\(\)](#) commit all changes in database

[rollbackTransaction\(\)](#) - rollback all changes in database

Sample code:

```
try{
    $this->beginTransaction();
    ... do something with data...
    $this->commitTransaction();
} catch (Exception $e){
    ... do something if wrong....
    $this->rollbackTransaction();
}
```

if you want class function called directly from javascript code in a Form, your function has to have arguments.

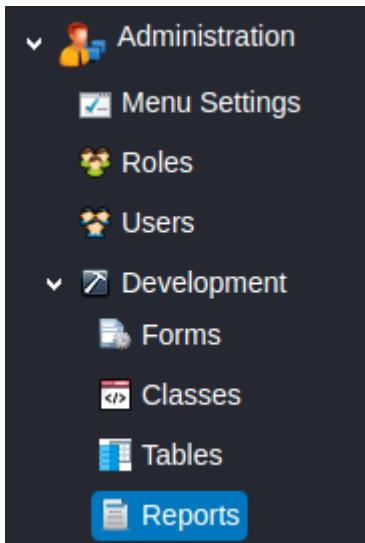
```
<?php
class post_gl_journal extends baseclass{
    function post_gl_recs($args=array()){
```

Call this function from Javascript code in button onClick function:

```
var buttons = [ {name:'post',label:'Post Journal',
    onClick:function(){
        if(!confirm("Post Gen. Journal Lines?")) return;
        var result = rcf('post_gl_journal','post_gl_recs',[maindata.formState.getSelectedRowsIndexes()]);
```

(Run Class Function) function **rcf**(<class name>,<function name>, array of arguments)

Reports



Administration → Development → Reports

in opened form add new record. Write report name in field *Report Name*

Every Report needs HTML template. You may use any HTML Editor. For example Libre Office Writer (or MS Word).

Divide your template on number of sections you need. Use double square quotes.

ration Serif 12 a a a a a^b a_b I_x a

1

[[purchase_quote_header_begin]]

Purchase Quote {{quote_number}}

| | |
|--|---|
| Buy from: {{buy_vendor_name}} {{buy_vendor_address}} Pay to: {{pay_vendor_name}} {{pay_vendor_address}} | {{company_name}} {{company_address}} |
|--|---|

2

| Type | Description | Quantity | Unit of Mes. | Price | Amount |
|---|--|---|--|--|---|
| [[purchase_quote_line_begin]] {{type}} | [[purchase_quote_line_begin]] {{Description}} | [[purchase_quote_line_begin]] {{Quantity}} | [[purchase_quote_line_begin]] {{uom}} | [[purchase_quote_line_begin]] {{price}} | [[purchase_quote_line_begin]] {{amount}} |
| [[purchase_quote_line_end]] | | | | | |

3

| | | | | | |
|---------------------------------|--|--|--|--------------|------------------|
| [[purchase_quote_footer_begin]] | | | | Total | {{total_amount}} |
|---------------------------------|--|--|--|--------------|------------------|

Shipment method: {{shipment_method}}
Ship to address: {{ship_to_address}}

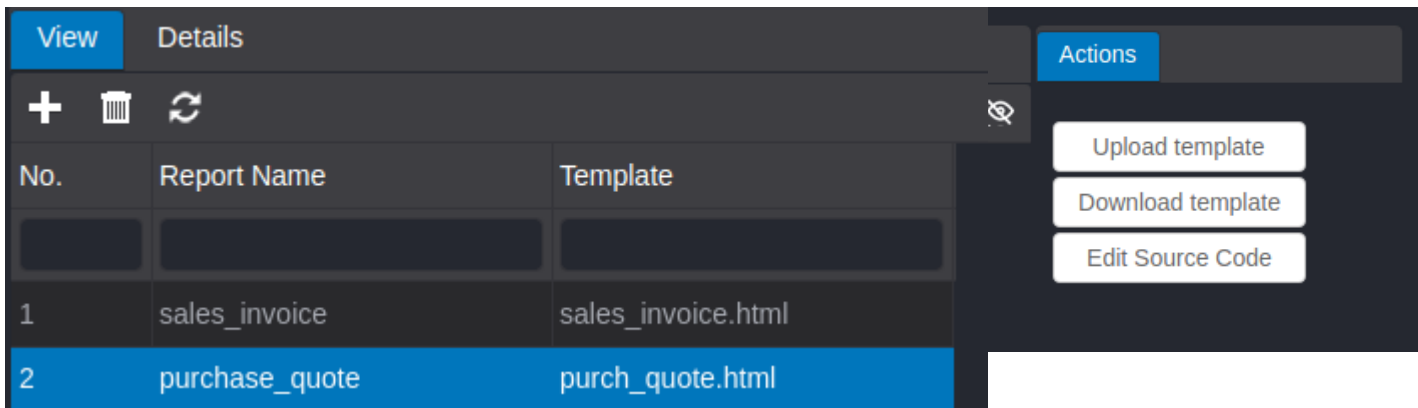
[[purchase_quote_footer_end]]

[[<section name>_begin]] ... section text ...[[<section name>_end]]
in section text place variables in double braces. {{<variable name>}}
Report has 3 sections:
purchase_quote_header, purchase_quote_line, purchase_quote_footer

after you finished with template Save As html.

Click button **Upload Template**

Choose the file and click OK to upload your template on server.



The screenshot shows a user interface for managing templates. It features a table with columns for 'No.', 'Report Name', and 'Template'. The second row is highlighted in blue. To the right of the table is an 'Actions' menu with three buttons: 'Upload template', 'Download template', and 'Edit Source Code'.

| No. | Report Name | Template |
|-----|----------------|--------------------|
| 1 | sales_invoice | sales_invoice.html |
| 2 | purchase_quote | purch_quote.html |

Actions:

- Upload template
- Download template
- Edit Source Code

If you need to edit template, you can download it by clicking **Download template**

After click button **Edit Source Code**

```

Edit Source Code purchase_quote
Save
4
5 ..... function build($args=array()){
6
7 ..... $purch_h = $args[0];
8 ..... $purchase_header=$this->getTable('purchase_header');
9 ..... $purchase_header->getById($purch_h);
10 ..... $company = $this->getTable('company_info');
11 ..... $company->getFirstRow();
12 ..... $sarr=array('quote_number'=>$purchase_header->get_Document_no(),
13 ..... 'buy_vendor_name'=>$purchase_header->obj_Buy_from_vendor_no()->get_Name(),
14 ..... 'buy_vendor_address'=>$purchase_header->obj_Buy_from_vendor_no()->get_Address(),
15 ..... 'pay_vendor_name'=>$purchase_header->obj_Pay_to_vendor_no()->get_Name(),
16 ..... 'pay_vendor_address'=>$purchase_header->obj_Pay_to_vendor_no()->get_Address()
17 ..... );
18 .....
19 ..... $sarr = array_merge($sarr,array(
20 ..... 'company_name'=>$company->get_Name(),
21 ..... 'company_address'=>$company->get_Address(),
22 ..... 'company_phone'=>$company->get_Phone(),
23 ..... 'company_email'=>$company->get_Email(),
24 ..... ));
25 ..... echo $this->buildSection('purchase_quote_header',$sarr);
26 .....
27 ..... $purchase_line = $this->getTable('purchase_line');
28 ..... $purchase_line->setFilter('header','',$purch_h);
29 ..... $purchase_line->setFilter('line amount','>',0);
30 ..... $purch_lines=$purchase_line->fetchAll();
31 ..... $order_line_types=$this->getTable('order_line_types');
32 ..... $order_line_types = $order_line_types->fetchHashMap();
33 ..... $uom = $this->getTable('unitofmeasure');
34 ..... $uom = $uom->fetchHashMap();
35 ..... $total_amount = 0;
36 ..... foreach($purch_lines as $rec){
37 ..... $sarr_line = array(
38 ..... 'type'=>$order_line_types[$rec['type']]['code'],
39 ..... 'Description'=>$rec['description'],
40 ..... 'Quantity'=>$rec['quantity'],
41 ..... 'uom'=>$uom[$rec['unit_of_measure']]['code'],
42 ..... 'price'=>$rec['unit_price'],
43 ..... 'amount'=>$rec['line_amount']
44 ..... );

```

Programming language is PHP

Extend your class from Report.

Override function **build**

Fill array with values. To print section purchase_quote_header fill array with values from double braces in your template.

```

$sarr = array(`quote_number`=>...,
    `buy_vendor_name`=>...,
    `buy_vendor_address`=>...,
    ....
);

```

print section with filled values:

```

echo $this->buildSection(<section name>, <array with values>);
section name is how you called it in your template file in double square quotes.

```

In form add button to call report and print it:

```
{name:'print', label:'Print', onClick:function(){
.....var result = rr('purchase_quote',[maindata.formState.rowdata.id]);
.....if(result!=''){
.....print(result);

}

}},
```

(Run Report) function **rr(<report name>, <arguments>)**
arguments will be passed in function **build** of your report.

You may see this example report in **Main Menu → Purchases & Payables → Documents → Quotes**