Active Record (AR)

AR is one of the most popular ways to accessing data in a relational database.

The general principles of operation of the Active Record is as follows:

A database table or view is wrapped into a class. Thus, an object instance is tied to a single row in the table. After creation of an object, a new row is added to the table upon save. Any object loaded gets its information from the database. When an object is updated the corresponding row in the table is also updated. The wrapper class implements accessor methods or properties for each column in the table or view.

In the ClickBlocks framework Active Record pattern is implemented by **ClickBlocks\DB\AR** class.

Creating AR Class

To access a database table, we need to create an instance of the AR class. The following example shows the simplest way to do it:

```
$ar = new AR('my_table');
```

The first argument of the AR constructor is a table name which gets associated with the created AR object. In the example above, the connection with a database is established via the global variable **db** that defined by **CB::set()** method. In this case the more detailed example could be as follows:

```
// Establishing database connection.
$db = new DB($dsn, $username, $password);
// Setting the global variable db.
CB::set('db', $db);
// Creating an AR instance.
$ar = new AR('my_table');
```

The second way to specify a database connection object is to define the static property of AR class:

```
// Establishing database connection.
$db = new DB($dsn, $username, $password);
// Defining the static property.
AR::$connection = $db;
// Creating an AR instance.
$ar = new AR('my_table');
```

Besides of setting of the global variable of the database connection and defining of the AR static property you can directly pass the DB instance as the second argument of the AR constructor:

```
// Establishing database connection.
$db = new DB($dsn, $username, $password);
// Creating an AR instance.
$ar = new AR('my_table', $db);
```

> If you established a database connection object by all three ways then the third way will have maximum priority against the rest two and the first way will have the minimum priority.

Creating Record

To insert a new row in a database table you need to create an instance of AR class, then set the properties associated with the table columns and, finally, call the **insert()** method. For example, consider the following database table (for MySQL DBMS):

```
CREATE TABLE `Users` (
  `userID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `typeID` int(10) unsigned NOT NULL,
  `firstName` varchar(100) NOT NULL,
  `lastName` varchar(100) NOT NULL,
  `email` varchar(500) NOT NULL,
```

```
  `password` varchar(100) NOT NULL,
  PRIMARY KEY (`userID`),
  KEY `kTypeID` (`typeID`),
  CONSTRAINT `fkTypeID` FOREIGN KEY (`typeID`)
                        REFERENCES LookupUserTypes (`typeID`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

By using AR class we can insert a new row in this table as follows:

```
// Creating a database connection object.
$db = new DB($dsn, $username, $password);
// Creating an AR instance.
$ar = new AR('Users', $db);
// Setting values of the AR properties.
$ar->typeID = 'U';
$ar->firstName = 'Frank';
$ar->lastName = 'Smith';
$ar->email = 'frank_smith@gmail.com';
$ar->password = 'admin';
// Inserting a new table row.
$ar->insert();
```

If the table's primary key is auto-incremental, then after the insertion the relative property will contain the value of the primary key of the latest inserted row. In the above example the property **$userID** contains the primary key value of the newly inserted user.

Moreover, the **insert()** method also returns the ID of the last inserted row, or the last value from a sequence object, depending on the underlying driver. For example, auto-incremental keys don't exist in Oracle or MSSQL DBMS. Iin these cases you need to pass in this method the sequence name:

```
// Creating an AR instance.
$ar = new AR('Users', $db);
// Setting values of the AR properties.
$ar->typeID = 'U';
...
// Inserting a new table row.
$userID = $ar->insert(['sequenceName' => 'seqUserID']);
```

> If a table doesn't have the auto-incremental column or the sequence name is not passed as a parameter, the method returns NULL.

We can assign SQL fragments to the AR properties as values. To do this we need to wrap the SQL fragment by **ClickBlocks\DB\SQLExpression** class or use the **exp()** method of AR class. For example, to save a MD5 hash of password returned by the MySQL MD5() function, we can use the following code:

```
// Creating an AR instance.
$ar = new AR('Users', $db);
// Setting values of the AR properties.
...
$ar->password = $ar->exp("MD5('admin')"); // new SQLExpression("MD5('admin')")
...
// Inserting a new table row.
$ar->insert();
```

> When we create the empty AR object all its properties get the default values of the appropriate table columns.

Reading Record

AR class has three methods for reading records from a database table which associated with the AR instance:

- `assign($where, $order = null)` - finds the first record according to the given criteria in the AR's table and assigns column values to the properties of the AR object.
- `select($columns = '*', $where = null, $order = null, $limit = null, $offset = null)` - finds records in the AR's table according to the given conditions.
- `count($where = null)` - returns number of the records in the AR's table according to the given condition.

The parameters of these methods corresponds to the parameters of the methods **select()**, **where()**, **order()** and **limit()** of the SQLBuilder class.

```
// Creating an AR instance.
$ar = new AR('Users', $db);

// Finds a row by its primary key value and assigns column values
// to the AR's properties.
$ar->assign(12);
// Finds a row according to more complex criteria.
$ar->assign(['typeID' => 'C', 'firstName' => 'Frank'], ['typeID' => 'DESC']);
// This is equivalent of the following SQL statement:
// SELECT * FROM `Users` WHERE `typeID` = 'C' AND `firstName` = 'Frank'
// ORDER BY `typeID` DESC LIMIT 1

// Finds several rows from the table.
$rows = $ar->select(['userID', 'typeID' => 'type', 'email'], [['>', 'userID', 10]], ['firstName'], 30);
// The above code performs SQL query like the following:
// SELECT `userID`, `typeID` `type`, `email` FROM `Users` WHERE `userID` > 10 ORDER BY `firstName` ASC LIMIT 30

// Finds number of the records corresponding the given criteria.
$count = $ar->count([['LIKE', 'email', 'frank_%']]);
```

Besides of the mentioned three methods the AR class has another one that allows to initialize the class properties with array values. This method is also useful when you need simultaneously to set values to several properties and initialize the AR object.

```
// Creating an AR instance.
$ar = new AR('Users', $db);
// Initializing properties from array.
$vs = ['firstName' => 'Frank', 'lastName' => 'Smith', 'typeID' => 'U'];
$ar->assignFromArray($vs);
```

In the above example each array element value is assigned to the class property with the same name. At that, if some array element has the key which doesn't match any property name then such element is ignored.

> Unlike the method **setValues()** that also can be used for setting values to several properties at once, the **assignFromArray()** method initializes AR object so as if the property values were get from a database. It is important for operation of the **save()** method (see below).

Updating Record

After an AR instance is populated with column values, we can change them and update them in the database via method **update()**:

```
// Creating an AR instance.
$ar = new AR('Users', $db);
// Initializing the AR instance with columns values.
$ar->assign(1);
// Changing properties' values.
$ar->typeID = 'C';
$ar->email = 'admin@gmail.com';
// Updating column values in the database.
$ar->update();
```

The **update()** method returns the number of affected rows.

There is one more method that allows to insert or update a rows in a database table. This is method **save()**. The main feature of this method is the automatic detection operation (insertion or updating) to be performed. If an AR object was initiated from a database table (and a row exists in this table) then the **save()** method will update a row after any changes, otherwise it will insert new row in the table.

```
// Creating an AR instance.
$ar = new AR('Users', $db);
// Populates the AR object with column values.
$ar->assign(2);
// Changes any column.
$ar->lastName = 'Finch';
```

```
// Updating the row.
$ar->save();

// Cleans properties' values and resets the internal states of the AR instance.
$ar->reset();

// Setting values of the AR properties.
$ar->typeID = 'U';
$ar->firstName = 'Frank';
$ar->lastName = 'Smith';
$ar->email = 'frank_smith@gmail.com';
$ar->password = 'admin';
// Inserting a new table row.
$userID = $ar->save();
```

When the **save()** method updates a row it is returning the number of affected rows and when it inserts a new table row the returning value is the same as the returning value of the **insert()** method.

You can also update some rows in a table by using **update()** method. To do this you need just to pass search conditions in this method:

```
// Creating an AR instance.
$ar = new AR('Users', $db);
// Setting property values.
$ar->setValues([...]);
// Updating several rows.
$ar->update(['typeID' => 'U']);  // It will update all rows with typeID = 'U'
```

> Note that for updating several rows you don't need initiate an AR instance with **assign()** method.

Deleting Record

You can also delete a row if an AR instance has been initiated with this row.

```
// Creating an AR instance.
$ar = new AR('Users', $db);
// Initiating of the AR object.
$ar->assign(1);
// Deleting the row.
$ar->delete();
```

The **delete()** method returns the number of affected rows. After deleting the row you won't be able to operate with this instance again (until you invoke **reset()** or **assign()** methods).

As in the case with **update()** method you can delete several rows from the database table.

```
// Creating an AR instance.
$ar = new AR('Users', $db);
// Deleting several rows.
$ar->delete(['typeID' => 'C']); // It will delete all rows with typeID = 'C'
```

> Note that for deleting several rows you don't need initiate an AR instance with **assign()** method.

Foreign Key Relationships

The AR class allows to get data from the relational tables via the specified foreign keys of the table. You can use method **relation()** for this purpose. For example, the Users table has the foreign key "fkTypeID". We can get the relative data as follows:

```
// Creating an AR instance.
$ar = new AR('Users', $db);
// Gets relative data.
$data = $ar->relation('fkTypeID');
// Also you can get part of relative data.
$data = $ar->relation('fkTypeID', ['type'], ['typeID' => 'DESC'], 5, 3);
```

> You can also use the index number instead of the foreign key name. But the recommended way is to use the foreign key names.

Caching Column Metadata

When an AR object is created the column metadata can be cached. You can mange the caching of table metadata via the static properties **$cacheExpire** and **$cacheGroup** which determines the cache lifetime and cache group respectively. Moreover you can define these parameters for each AR instance by passing them to the AR constructor:

```
// Sets the default cache parameters for all instances of the AR class.
AR::$cacheExpire = 36000;
AR::$cacheGroup = 'ar';
// Creating an AR instance and setting the individual cache parameters.
$ar = new AR('Users', $db, $cacheExpire, $cacheGroup);
```

If you didn't set the default cache parameters via the static properties or didn't pass them to the AR constructor, the default cache parameters will be taken from the appropriate configuration setting from the **ar** section.

```
[ar]
cacheExpire = 9000
cacheGroup  = 'ar'
```

> If you want to turn off the caching of a table metadata you need to set **cacheExpire** to 0 or FALSE. If you set **cacheExpire** to a value that less than zero, the cache lifetime will be equal to the database cache vault lifetime.