

mySQL DAO

Generated by Doxygen 1.8.6

Thu Jul 3 2014 12:03:12

Contents

1	Hierarchical Index	1
1.1	Class Hierarchy	1
2	Class Index	3
2.1	Class List	3
3	Class Documentation	5
3.1	Address Class Reference	5
3.1.1	Detailed Description	6
3.1.2	Constructor & Destructor Documentation	6
3.1.2.1	__construct	6
3.1.3	Member Function Documentation	7
3.1.3.1	getCity	7
3.1.3.2	getId	7
3.1.3.3	getName	7
3.1.3.4	getState	7
3.1.3.5	getStreet1	7
3.1.3.6	getStreet2	8
3.1.3.7	getZip	8
3.1.3.8	setCity	8
3.1.3.9	setId	8
3.1.3.10	setName	8
3.1.3.11	setState	8
3.1.3.12	setStreet1	9
3.1.3.13	setStreet2	9
3.1.3.14	setZip	9
3.1.4	Member Data Documentation	9
3.1.4.1	\$dao_typeMap	9
3.2	MySQLDAO Class Reference	10
3.2.1	Detailed Description	10
3.2.2	Member Function Documentation	11
3.2.2.1	dao_generateBindParameters	11

3.2.2.2	dao_generateFieldList	11
3.2.2.3	dao_generateParameters	11
3.2.2.4	dao_generateUpdateParameters	11
3.2.2.5	delete	12
3.2.2.6	insert	12
3.2.2.7	update	12
Index		13

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

MySQLDAO	10
Address	5

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Address	Example class demonstrating the use of MySQLDAO	5
MySQLDAO	Abstract class for the mySQL DAO design pattern	10

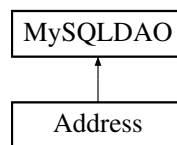
Chapter 3

Class Documentation

3.1 Address Class Reference

example class demonstrating the use of [MySQLDAO](#)

Inheritance diagram for Address:



Public Member Functions

- [__construct](#) (\$newId, \$newName, \$newStreet1, \$newStreet2, \$newCity, \$newState, \$newZip)
constructor for [Address](#)
- [getId](#) ()
accessor method for id
- [setId](#) (\$newId)
mutator method for id
- [getName](#) ()
accessor method for name
- [setName](#) (\$newName)
mutator method for name
- [getStreet1](#) ()
accessor method for first line of street address
- [setStreet1](#) (\$newStreet1)
mutator method for first line of street address
- [getStreet2](#) ()
accessor method for second line of street address
- [setStreet2](#) (\$newStreet2)
mutator method for second line of street address
- [getCity](#) ()
accessor method for city
- [setCity](#) (\$newCity)
mutator method for city
- [getState](#) ()

- accessor method for state*
- [setState](#) (\$newState)
mutator method for state
- [getZip](#) ()
accessor method for zip
- [setZip](#) (\$newZip)
mutator method for zip

Protected Attributes

- [\\$id](#)
id of the address; this is the primary key
- [\\$name](#)
name of the person or company
- [\\$street1](#)
first line of the street address
- [\\$street2](#)
second line of the street address
- [\\$city](#)
city of the street address
- [\\$state](#)
state of the street address
- [\\$zip](#)
ZIP code of the street address.

Static Protected Attributes

- static [\\$dao_primaryKey](#) = "id"
MySQLDAO's name of the primary key.
- static [\\$dao_tableName](#) = "address"
MySQLDAO's name of the table.
- static [\\$dao_typeMap](#)
MySQLDAO's map of the data types.

Additional Inherited Members

3.1.1 Detailed Description

example class demonstrating the use of [MySQLDAO](#)

This is a typical example of a class that represents a row in a mySQL table. Notice the class is simple and clean and consists of only typical accessor and mutator methods. All one needs to do is extend the [MySQLDAO](#) class and include the static DAO related variables to deploy the DAO pattern.

Author

Dylan McDonald dylanm@deepdivecoders.com

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Address::__construct (\$newId, \$newName, \$newStreet1, \$newStreet2, \$newCity, \$newState, \$newZip)

constructor for [Address](#)

Parameters

<i>integer</i>	new value of id
<i>string</i>	new value of name
<i>string</i>	new value of first line of street address
<i>string</i>	new value of second line of street address
<i>string</i>	new value of city
<i>string</i>	new value of state
<i>string</i>	new value of zip

Exceptions

<i>RuntimeException</i>	if invalid inputs detected
-------------------------	----------------------------

3.1.3 Member Function Documentation

3.1.3.1 Address::getCity ()

accessor method for city

Returns

string value of city

3.1.3.2 Address::getId ()

accessor method for id

Returns

integer value of id

3.1.3.3 Address::getName ()

accessor method for name

Returns

string value of name

3.1.3.4 Address::getState ()

accessor method for state

Returns

string value of state

3.1.3.5 Address::getStreet1 ()

accessor method for first line of street address

Returns

string value of first line of street address

3.1.3.6 Address::getStreet2 ()

accessor method for second line of street address

Returns

string value of second line of street address

3.1.3.7 Address::getZip ()

accessor method for zip

Returns

string value of zip

3.1.3.8 Address::setCity (\$newCity)

mutator method for city

Parameters

<i>string</i>	new value of city
---------------	-------------------

Exceptions

<i>UnexpectedValueException</i>	if location is empty
---------------------------------	----------------------

3.1.3.9 Address::setId (\$newId)

mutator method for id

Parameters

<i>integer</i>	new value of id
----------------	-----------------

Exceptions

<i>RangeException</i>	if id is negative
<i>UnexpectedValueException</i>	if id is invalid

3.1.3.10 Address::setName (\$newName)

mutator method for name

Parameters

<i>string</i>	new value of name
---------------	-------------------

Exceptions

<i>UnexpectedValueException</i>	if name is empty
---------------------------------	------------------

3.1.3.11 Address::setState (\$newState)

mutator method for state

Parameters

<i>string</i>	new value of state
---------------	--------------------

Exceptions

<i>UnexpectedValueException</i>	if state is invalid
---------------------------------	---------------------

3.1.3.12 Address::setStreet1 (*\$newStreet1*)

mutator method for first line of street address

Parameters

<i>string</i>	new value of first line of street address
---------------	---

Exceptions

<i>UnexpectedValueException</i>	if location is empty
---------------------------------	----------------------

3.1.3.13 Address::setStreet2 (*\$newStreet2*)

mutator method for second line of street address

Parameters

<i>string</i>	new value of second line of street address
---------------	--

3.1.3.14 Address::setZip (*\$newZip*)

mutator method for zip

Parameters

<i>string</i>	new value of zip
---------------	------------------

Exceptions

<i>UnexpectedValueException</i>	if zip is invalid
---------------------------------	-------------------

3.1.4 Member Data Documentation

3.1.4.1 Address::\$dao_typeMap [static], [protected]

Initial value:

```
= array("id" => "i", "name" => "s", "street1" => "s", "street2" => "s",
        "city" => "s", "state" => "s", "zip" => "s")
```

[MySQLDAO](#)'s map of the data types.

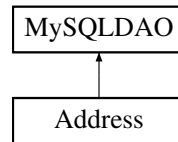
The documentation for this class was generated from the following file:

- address.php

3.2 MySQLDAO Class Reference

abstract class for the mySQL DAO design pattern

Inheritance diagram for MySQLDAO:



Public Member Functions

- **delete** (&\$mysqli)
deletes this object from mySQL
- **insert** (&\$mysqli)
inserts this object to mySQL
- **update** (&\$mysqli)
updates this object in mySQL

Protected Member Functions

- **dao_generateBindParameters** (\$fieldList)
generates an array to pass to mysqli_stmt::bind_param()
- **dao_generateFieldList** (\$omitPrimaryKey)
generates a comma separated field list
- **dao_generateParameters** (\$numFields)
generates placeholders for an INSERT query template
- **dao_generateUpdateParameters** (\$fieldList)
generates placeholders for an UPDATE query template

3.2.1 Detailed Description

abstract class for the mySQL DAO design pattern

This class is the abstract class for the mySQL Data Access Object (DAO) pattern. Each subclass of this class will receive the following public methods:

- delete(&\$mysqli);
- insert(&\$mysqli);
- update(&\$mysqli);

In order to use the DAO pattern, subclasses must have the following protected static variables:

- dao_primaryKey: string containing the primary key's field name
- dao_tableName: string containing the mySQL table name
- dao_typeMap: array containing "field" => "type" pairs, where field is the field name and type is the data type according to mysqli_stmt::bind_param().

Author

Dylan McDonald dylanm@deepdivecoders.com

See Also

<http://www.php.net/manual/en/mysqli-stmt.bind-param.php>

3.2.2 Member Function Documentation

3.2.2.1 MySQLDAO::dao_generateBindParameters (*\$fieldList*) [protected]

generates an array to pass to mysqli_stmt::bind_param()

Parameters

<i>string</i>	comma separated field list
---------------	----------------------------

Returns

array arguments to pass to mysqli_stmt::bind_param()

See Also

<http://www.php.net/manual/en/mysqli-stmt.bind-param.php>

3.2.2.2 MySQLDAO::dao_generateFieldList (*\$omitPrimaryKey*) [protected]

generates a comma separated field list

Parameters

<i>boolean</i>	whether to omit the primary key
----------------	---------------------------------

Returns

string comma separated field list

3.2.2.3 MySQLDAO::dao_generateParameters (*\$numFields*) [protected]

generates placeholders for an INSERT query template

Parameters

<i>integer</i>	number of fields to template
----------------	------------------------------

Returns

string templated placeholders

3.2.2.4 MySQLDAO::dao_generateUpdateParameters (*\$fieldList*) [protected]

generates placeholders for an UPDATE query template

Parameters

<i>string</i>	comma separated field list
---------------	----------------------------

Returns

string templated placeholders

3.2.2.5 MySQLDAO::delete (& \$mysqli)

deletes this object from mySQL

Parameters

<i>mysqli</i>	pointer to valid mysqli connection
---------------	------------------------------------

Exceptions

<i>mysqli_sql_exception</i>	if a mySQL error occurs
-----------------------------	-------------------------

3.2.2.6 MySQLDAO::insert (& \$mysqli)

inserts this object to mySQL

Parameters

<i>mysqli</i>	pointer to valid mysqli connection
---------------	------------------------------------

Exceptions

<i>mysqli_sql_exception</i>	if a mySQL error occurs
-----------------------------	-------------------------

3.2.2.7 MySQLDAO::update (& \$mysqli)

updates this object in mySQL

Parameters

<i>mysqli</i>	pointer to valid mysqli connection
---------------	------------------------------------

Exceptions

<i>mysqli_sql_exception</i>	if a mySQL error occurs
-----------------------------	-------------------------

The documentation for this class was generated from the following file:

- mysqldao.php

Index

\$dao_typeMap
 Address, 9
__construct
 Address, 6

Address, 5
 \$dao_typeMap, 9
 __construct, 6
 getCity, 7
 getId, 7
 getName, 7
 getState, 7
 getStreet1, 7
 getStreet2, 7
 getZip, 8
 setCity, 8
 setId, 8
 setName, 8
 setState, 8
 setStreet1, 9
 setStreet2, 9
 setZip, 9

dao_generateBindParameters
 MySQLDAO, 11
dao_generateFieldList
 MySQLDAO, 11
dao_generateParameters
 MySQLDAO, 11
dao_generateUpdateParameters
 MySQLDAO, 11
delete
 MySQLDAO, 12

getCity
 Address, 7
getId
 Address, 7
getName
 Address, 7
getState
 Address, 7
getStreet1
 Address, 7
getStreet2
 Address, 7
getZip
 Address, 8

insert

MySQLDAO, 12

MySQLDAO, 10
 dao_generateBindParameters, 11
 dao_generateFieldList, 11
 dao_generateParameters, 11
 dao_generateUpdateParameters, 11
 delete, 12
 insert, 12
 update, 12

setCity
 Address, 8
setId
 Address, 8
setName
 Address, 8
setState
 Address, 8
setStreet1
 Address, 9
setStreet2
 Address, 9
setZip
 Address, 9

update
 MySQLDAO, 12