ClickBlocks\DB\SQLBuilder

General information

Inheritance	no
Child classes	ClickBlocks\DB\MySQLBuilder, ClickBlocks\DB\SQLiteBuilder, ClickBlocks\DB\OCIBuilder
Interfaces	no
Source	Framework/db/core/sqlbuilder.php

SQLBuilder is the base abstract class for all classes that intended for building SQL queries depending on DBMS type. All its child classes have the same methods.

Public static methods

getInstance()

```
public static ClickBlocks\DB\SQLBuilder getInstance(string $engine)
```

$engine	string	the database engine.

Returns an instance of the builder class corresponding to the given database engine. At present you can create builder class for three database engine: MySQL, SQLite and Oracle. If the builder class for the given database engine does not exist the method will throw exception with token `ClickBlocks\DB\SQLBuilder::ERR_SQL_1`.

Public non-static methods

__construct()

```
public void __construct(ClickBlocks\DB\DB $db = null)
```

$db	ClickBlocks\DB\DB	the database connection object.

The class constructor that stores the database connection objects in the appropriate internal class property.

exp()

```
public ClickBlocks\DB\SQLExpression exp(string $sql)
```

$sql	string	any SQL expression.

Converts any string to `ClickBlocks\DB\SQLExpression` object.

getPHPType()

```
abstract public string getPHPType(string $type)
```

$type	string	the SQL data type.

Converts the given SQL data type of the particular DBMS to PHP type.

wrap()

```
abstract public string wrap(string $name, boolean $isTableName = false)
```

$name	string	the column or table name.
$isTableName	boolean	determines whether **$name** is a table name.

Quotes a table or column name for use in SQL queries.

quote()

```
abstract public string|array quote(string|array $value, string $format = ClickBlocks\DB\SQLBuilder::ESCAPE_QUOTED_VALUE)
```

$value	string, array	any string or an array of string values.
$format	string	determines the format of the quoted value. This value must be one of the `ClickBlocks\DB\SQLBuilder::ESCAPE_*` constants.

Quotes a value (or an array of values) to produce a result that can be used as a properly escaped data value in an SQL statement. if **$value** is an array then all its elements will be quoted.

tableList()

```
abstract public array tableList(string $scheme = null)
```

$scheme	string	the table scheme (database name).

Returns SQL for getting the table list of the current database.

tableInfo()

```
abstract public string tableInfo(string $table)
```

$table	string	the table name.

Returns SQL for getting metadata of the specified table.

columnsInfo()

```
abstract public string columnsInfo(string $table)
```

$table	string	the table name.

Returns SQL for getting metadata of the table columns.

createTable()

```
abstract public string createTable(string $table, array $columns, array $options = null)
```

$table	string	the name of the table to be created.

$columns	array	the columns information of the new table.
$options	array	additional SQL fragment that will be appended to the generated SQL.

Returns SQL for creating a new DB table.

renameTable()

```
abstract public string renameTable(string $oldName, string $newName)
```

$oldName	string	the old table name.
$newName	string	the new table name.

Returns SQL for renaming a table.

dropTable()

```
abstract public string dropTable(string $table)
```

$table	string	the table name.

Returns SQL that can be used for removing the particular table.

truncateTable()

```
abstract public string truncateTable(string $table)
```

$table	string	the table name.

Returns SQL that can be used to remove all data from a table.

addColumn()

```
abstract public string addColumn(string $table, string $column, string $type)
```

$table	string	the table that the new column will be added to.
$column	string	the name of the new column.
$type	string	the DBMS-dependent column type.

Returns SQL for adding a new column to a table.

renameColumn()

```
abstract public string renameColumn(string $table, string $oldName, string $newName)
```

$table	string	the table whose column is to be renamed.
$oldName	string	the previous name of the column.
$newName	string	the new name of the column.

Returns SQL for renaming a column.

changeColumn()

```
abstract public string changeColumn(string $table, string $oldName, string $newName, string $type)
```

$table	string	the table whose column is to be changed.
$oldName	string	the old name of the column.
$newName	string	the new name of the column.
$type	string	the DBMS-dependent type of the column.

Returns SQL for changing the definition of a column.

dropColumn()

```
abstract public string dropColumn(string $table, string $column)
```

$table	string	the table whose column is to be dropped.
$column	string	the name of the column to be dropped.

Returns SQL for dropping a DB column.

addForeignKey()

```
abstract public string addForeignKey(string $name, string $table, array $columns, string $refTable, array $refColumns, string $del
```

$name	string	the name of the foreign key constraint.
$table	string	the table that the foreign key constraint will be added to.
$columns	array	the column(s) to that the constraint will be added on.
$refTable	string	the table that the foreign key references to.
$refColumns	array	the column(s) that the foreign key references to.
$delete	string	the ON DELETE option. Most DBMS support these options: RESTRICT, CASCADE, NO ACTION, SET DEFAULT, SET NULL.
$update	string	the ON UPDATE option. Most DBMS support these options: RESTRICT, CASCADE, NO ACTION, SET DEFAULT, SET NULL.

Returns SQL for adding a foreign key constraint to an existing table.

dropForeignKey()

```
abstract public string dropForeignKey(string $name, string $table)
```

$name	string	the name of the foreign key constraint to be dropped.
$table	string	the table whose foreign key is to be dropped.

Returns SQL for dropping a foreign key constraint.

createIndex()

```
abstract public string createIndex(string $name, string $table, array $columns, string $class = null, string $type = null)
```

$name	string	the index name.
$table	string	the table that the new index will be created for.
$columns	array	the columns that should be included in the index.
$class	string	the index class. For example, it can be UNIQUE, FULLTEXT and etc.
$type	string	the DBMS-dependent index type.

Returns SQL for creating a new index.

dropIndex()

```
abstract public string dropIndex(string $name, string $table)
```

$name	string	the name of the index to be dropped.
$table	string	the table whose index is to be dropped.

Returns SQL for dropping an index.

normalizeColumnsInfo()

```
abstract public array normalizeColumnsInfo(array $info)
```

$info	array	the columns metadata.

Normalizes the metadata of the DB columns. The method returns the canonized information about columns of the given table. This information has common structure for all DBMS:

Array key name	Type	Description
column	string	the column name.
type	string	the database column type.
phpType	string	the PHP column data type.
isPrimaryKey	boolean	determines whether the column is the part of primary key.
isNullable	boolean	determines whether the column allows NULL.
isAutoincrement	boolean	determines whether the column value can be automatically created.
isUnsigned	boolean	determines whether the column takes only positive values.
default	boolean	the default column value.
maxLength	boolean	the maximum length of the column value.
precision	boolean	the number of digits to the right of the decimal point.
set	array	the array values of column one of enumeration types.

normalizeTableInfo()

```
public array normalizeTableInfo(array $info)
```

$info	array	the table metadata received from the database.

Normalizes the database table metadata and returns the canonized information about the table constraints and keys. The structure of this information is common for all DBMS:

Array key name	Type	Description
constraints	array	the list of all table constraints.
keys	array	the list of all table keys.
engine	string	the table engine.
charset	string	the table default charset.
autoincrementInitialValue	integer	the initial value of the autoincrement column of the table.

insert()

```
public ClickBlocks\DB\SQLBuilder insert(string $table, mixed $columns, array $options = null)
```

$table	string	the table name.
$columns	mixed	the columns to insert.
$options	array	additional information about SQL for some DBMS.

Starts to form the SQL query of INSERT type.

$columns** can be a string, **ClickBlocks\DB\SQLExpression** instance, one-dimensional associative array and multi-dimensional associative array. If **$columns is a string then this string becomes the part of the final SQL without any changes.

$options can contains additional parameters specified for some DBMS. For example, for MySQL DBMS we can set boolean parameter **updateOnKeyDuplicate**. If this parameter is specified and a row is inserted that would cause a duplicate value in a UNIQUE index or PRIMARY KEY, an UPDATE of the old row is performed.

The examples below display the mapping between parameters of the method and final SQL string (for SQLite):

1. PHP => SQL

```
$sql->insert('MyTable', '1, 2, 3')
```

```
INSERT INTO "MyTable" (1, 2, 3)
```

2. PHP => SQL

```
$sql->insert('MyTable', ['c1' => 'v1', 'c2' => 'v2', 'c3' => 'v3'])
```

```
INSERT INTO "MyTable" ("c1", "c2", "c3") VALUES (?, ?, ?)
```

3. PHP => SQL

```
$sql->insert('tb', ['c1' => ['a', 'b'], 'c2' => 'foo', 'c3' => [1, 2, new SQLExpression('DATE(\'now\')')]])
```

```
INSERT INTO "tb" ("c1", "c2", "c3") VALUES (?, ?, ?), (?, ?, ?), (?, ?, DATE('now'))
```

update()

```
public ClickBlocks\DB\SQLBuilder update(mixed $table, mixed $columns, array $options = null)
```

$table	mixed	the table name(s).
$columns	mixed	the columns to update.
$options	array	additional information about SQL for some DBMS.

Starts to form the SQL query of UPDATE type.

$table can be a string, **ClickBlocks\DB\SQLExpression** instance or one-dimensional mixed array.

$columns** can be a string, **ClickBlocks\DB\SQLExpression** instance or one-dimensional associative array. If **$columns is a string then this string becomes the part of the final SQL without any changes.

The examples below display the mapping between parameters of the method and final SQL string (for SQLite):

1. PHP => SQL

```
$sql->update(new SQLExpression('tb AS t'), 'c1 = 1, c2 = c2 + 1')
```

```
UPDATE tb AS t SET c1 = 1, c2 = c2 + 1
```

2. PHP => SQL

```
$sql->update('tb', ['c1' => 'a', 'c2' => 'b', 'c3' => 'c'])
```

```
UPDATE "tb" SET "c1" = ?, "c2" = ?, "c3" = ?
```

3. PHP => SQL

```
$sql->update(['tb1', 'tb2' => 't2'], ['c1' => 1, new SQLExpression('c = c + 1'), 'c2' => 'abc'])
```

```
UPDATE "tb1", "tb2" "t2" SET "c1" = ?, c = c + 1, "c2" = ?
```

delete()

```
public ClickBlocks\DB\SQLBuilder delete(mixed $table, array $options = null)
```

$table	mixed	the table name(s).
$options	array	additional information about SQL for some DBMS.

Starts to form the SQL query of DELETE type.

$table can be a string, **ClickBlocks\DB\SQLExpression** instance or one-dimensional mixed array.

The examples below display the mapping between parameters of the method and final SQL string (for SQLite):

1. PHP => SQL

```
$sql->delete('tb')
```

```
DELETE FROM "tb"
```

2. PHP => SQL

```
$sql->delete(new SQLExpression('tb AS t'))
```

```
DELETE FROM tb AS t
```

3. PHP => SQL

```
$sql->delete(['tb1', 'tb2' => 't'])
```

```
DELETE FROM "tb1", "tb2" "t"
```

select()

```
public ClickBlocks\DB\SQLBuilder select(mixed $table, mixed $columns = '*', string $distinct = null, array $options = null)
```

$table	mixed	the table name(s).
$columns	mixed	the columns that supposed to obtain from the database.
$distinct	string	additional select options for some DBMS.
$options	array	additional query information for some DBMS.

Starts to form the SQL query of SELECT type.

$table can be a string, **ClickBlocks\DB\SQLExpression** instance or one-dimensional mixed array.

$columns** can be a string, **ClickBlocks\DB\SQLExpression** instance or one-dimensional mixed array. If **$columns is a string then this string becomes the part of the final SQL without any changes.

The examples below display the mapping between parameters of the method and final SQL string (for SQLite):

1. PHP => SQL

```
$sql->select('tb', 'c1, c2, c3')
```

```
SELECT c1, c2, c3 FROM "tb"
```

2. PHP => SQL

```
$sql->select(['tb' => 't'], ['c1', 'c2', 'c3'])
```

```
SELECT "c1", "c2", "c3" FROM "tb"
```

3. PHP => SQL

```
$sql->select(['tb1', 'tb2' => 't'], ['c1', 't.c2' => 'col', new SQLExpression('COUNT(c3) AS n')], 'DISTINCT')
```

```
SELECT DISTINCT "c1", "t"."c2" "col", COUNT(c3) AS n FROM "tb1", "tb2" "t"
```

join()

```
public ClickBlocks\DB\SQLBuilder join(mixed $table, mixed $conditions, string $type = 'INNER')
```

$table	mixed	the table(s) to join.
$conditions	mixed	the JOIN clause information.
$type	string	the JOIN clause type.

Applies JOIN clause to the current SQL query.

$table can be a string, **ClickBlocks\DB\SQLExpression** instance or one-dimensional mixed array.

$conditions can take values in the same format as method **where**.

The examples below display the mapping between parameters of the method and final SQL string (for SQLite):

1. PHP => SQL

```
$sql->select('tb1', '*')->join('tb2', 'tb2.col = tb1.col')
```

```
SELECT * FROM "tb1" INNER JOIN "tb2" ON tb2.col = tb1.col
```

2. PHP => SQL

```
$sql->select(['tb1' => 't1'], '*')
    ->join(['tb2' => 't2', 'tb3' => 't3'],
           [['or', ['=', 't3.col', new SQLExpression('t1.col')], ['=', 't3.col', new SQLExpression('t2.col')]],
            ['IS', 't1.ID', 'NOT NULL'],
            ['=', 't2', 1]], 'LEFT')
```

```
SELECT * FROM "tb1" "t1"
LEFT JOIN "tb2" "t2", "tb3" "t3" ON ("t3"."col" = t1.col OR "t3"."col" = t2.col) AND t1.ID IS NOT NULL AND "t2" = ?
```

where()

```
public ClickBlocks\DB\SQLBuilder where(mixed $conditions)
```

$conditions	mixed	the WHERE clause information.

Applies WHERE clause to the current SQL query.

$conditions can be a string, **ClickBlocks\DB\SQLExpression** instance or multi-dimensional mixed array.

The examples below display the mapping between parameters of the method and final SQL string (for SQLite):

1. PHP => SQL

```
$sql->select('tb', '*')->where('c1 > 1 AND c2 <> 2');
```

```
SELECT * FROM "tb" WHERE c1 > 1 AND c2 > 2
```

2. PHP => SQL

```
$sql->select('tb', '*')->where([['=', 'c1', 1], ['=', 'c2', 2], ['=', 'c3', '3']])
```

```
SELECT * FROM "tb" WHERE "c1" = ? AND "c2" = ? AND "c3" = ?
```

3. PHP => SQL

```
$sql->select('tb', '*')
    ->where([['=', 'c1', new SQLExpression('c2')],
             'c3 LIKE c4',
             ['or', ['=', 'c3', 3], ['c4', 'IN', [1, 2, 3]]]])
```

```
SELECT * FROM "tb" WHERE "c1" = c2 AND c3 LIKE c4 AND ("c3" = ? OR "c4" IN (?, ?, ?))
```

group()

```
public ClickBlocks\DB\SQLBuilder group(mixed $group)
```

$group	mixed	the GROUP clause information.

Applies GROUP clause to the current SQL query.

$group can be a string, **ClickBlocks\DB\SQLExpression** instance or one-dimensional numeric array.

The examples below display the mapping between parameters of the method and final SQL string (for SQLite):

1. PHP => SQL

```
$sql->select('tb', '*')->group('c1');
```

```
SELECT * FROM "tb" GROUP BY "c1"
```

2. PHP => SQL

```
$sql->select('tb', '*')->group(['c1', new SQLExpression('c2')]);
```

```
SELECT * FROM "tb" GROUP BY "c1", c2
```

having()

```
public ClickBlocks\DB\SQLBuilder having(mixed $conditions)
```

$conditions	mixed	the HAVING clause information.

Applies HAVING clause to the current SQL query.

$conditions can be a string, **ClickBlocks\DB\SQLExpression** instance or multi-dimensional mixed array.

The examples below display the mapping between parameters of the method and final SQL string (for SQLite):

1. PHP => SQL

```
$sql->select('tb', '*')->having('c1 = 1 OR c2 = 2')
```

```
SELECT * FROM "tb" HAVING c1 = 1 OR c2 = 2
```

2. PHP => SQL

```
$sql->select('tb', '*')
    ->having(['or', ['and', ['=', 'c1', 1], ['=', 'c2', 2]], ['IN', 'c3', [1, 2, 3]]])
```

```
SELECT * FROM "tb" HAVING "c1" = ? AND "c2" = ? OR "c3" IN (?, ?, ?)
```

order()

```
public ClickBlocks\DB\SQLBuilder order(mixed $order)
```

$order	mixed	the ORDER clause information.

Applies ORDER clause to the current SQL query.

$order can be a string, **ClickBlocks\DB\SQLExpression** instance or one-dimensional mixed array.

The examples below display the mapping between parameters of the method and final SQL string (for SQLite):

1. PHP => SQL

```
$sql->select('tb', '*')->order('c1, c2 DESC')
```

```
SELECT * FROM "tb" ORDER BY c1, c2 DESC
```

2. PHP => SQL

```
$sql->select('tb', '*')->order(['c1', 'c2' => 'DESC'])
```

```
SELECT * FROM "tb" ORDER BY "c1", "c2" DESC
```

limit()

```
public ClickBlocks\DB\SQLBuilder limit(integer $limit, integer $offset = null)
```

$limit	integer	the maximum number of rows.
$offset	integer	the row offset.

Applies LIMIT clause to the current SQL query.

build()

```
public string build(mixed &$data = null)
```

$data	mixed	a variable in which the data array for the SQL query will be written.

Returns completely formed SQL query of the given type.

Protected non-static properties

db

```
protected ClickBlocks\DB\DB $db
```

The database connection object.

sql

```
protected array $sql = []
```

Array of data for SQL building.

seq

```
protected integer $seq = 0
```

Determines whether the named (**`$seq`** > 0) or question mark placeholder (**`$seq`** is 0 or FALSE) are used in the SQL statement.

Protected non-static methods

buildInsert()

```
protected string buildInsert(array $insert, mixed &$data)
```

$insert	array	the query data.
$data	mixed	a variable in which the data array for the SQL query will be written.

Returns completed SQL query of INSERT type.

buildUpdate()

```
protected string buildUpdate(array $update, mixed &$data)
```

$update	array	the query data.
$data	mixed	a variable in which the data array for the SQL query will be written.

Returns completed SQL query of UPDATE type.

buildDelete()

```
protected string buildDelete(array $delete, mixed &$data)
```

$delete	array	the query data.
$data	mixed	a variable in which the data array for the SQL query will be written.

Returns completed SQL query of DELETE type.

buildSelect()

```
protected string buildSelect(array $select, mixed &$data)
```

$select	array	the query data.
$data	mixed	a variable in which the data array for the SQL query will be written.

Returns completed SQL query of SELECT type.

insertExpression()

```
protected array insertExpression(mixed $expression, mixed &$data)
```

$expression	mixed	the column information.
$data	mixed	a variable in which the data array for the SQL query will be written.

Normalizes the column information for the INSERT type SQL query.

updateExpression()

```
protected array updateExpression(mixed $expression, mixed &$data)
```

$expression	mixed	the column information.

$data	mixed	a variable in which the data array for the SQL query will be written.

Normalizes the column information for the UPDATE type SQL query.

selectExpression()

```
protected array selectExpression(mixed $expression, boolean $isTableName = false, boolean $isOrderExpression = false)
```

$expression	mixed	the column information.
$isTableName	boolean	determines whether **$expression** is a table name(s).
$isOrderExpression	boolean	determines whether **$expression** is an order expression or not.

Normalizes the column information for the SELECT type SQL query.

whereExpression()

```
protected array whereExpression(mixed $expression, mixed &$data, string $conjunction = null)
```

$expression	mixed	the column information.
$data	mixed	a variable in which the data array for the SQL query will be written.
$conjunction	string	conjunction SQL keyword of a WHERE clause.

Normalizes the column information for WHERE clause of the SQL query.

addParam()

```
protected string addParam(mixed $value, array &$data)
```

$value	mixed	the value to be added.
$data	array	the SQL statement parameters to which the value will be added.

Adds a parameter value to the SQL statement and returns the part of the building SQL string.