UnderQuery Documentation

Release 0.5.0

Beniamin Jonatan Šimko

December 22, 2016

Contents

Welcome to the UnderQuery documentation. Source code can be found on GitHub.

CHAPTER 1

Quick Start

1.1 Installation

```
composer require phuria/under-query
```

1.2 Entry point

```
$uq = new \Phuria\UnderQuery\UnderQuery();
```

CHAPTER 2

Examples

There are different query builder classes for each SQL query type: *SelectBuilder*, *UpdateBuilder*, *DeleteBuilder* and *InsertBuilder*. To create them we will use our factory:

```
$phuriaSQL = new \Phuria\SQLBuilder\PhuriaSQLBuilder();
```

2.1 Simple SELECT

```
$qb = $phuriaSQL->createSelect();

$qb->addSelect('u.name', 'c.phone_number');
$qb->from('user', 'u');
$qb->leftJoin('contact', 'c', 'u.id = c.user_id');

echo $qb->buildSQL();
```

2.2 Single table DELETE

```
$qb = $phuriaSQL->createDelete();

$qb->from('user');
$qb->andWhere('id = 1');

echo $qb->buildSQL();
```

```
DELETE FROM user WHERE id = 1;
```

2.3 Multiple table DELETE

```
$qb = $phuriaSQL->createDelete();

$qb->from('user', 'u');
$qb->innerJoin('contact', 'c', 'u.id = c.user_id')
$qb->addDelete('u', 'c');
$qb->andWhere('u.id = 1');
```

```
echo $qb->buildSQL();
```

```
DELETE u, c FROM user u LEFT JOIN contact c ON u.id = c.user_id WHERE u.id = 1
```

2.4 Simple INSERT

```
$qb = $phuriaSQL->createInsert();

$qb->into('user', 'u', ['username', 'email']);
$qb->addValues(['phuria', 'spam@simko.it']);

echo $qb->buildSQL();
```

```
INSERT INTO user (username, email) VALUES ("phuria", "spam@simko.it")
```

2.5 INSERT ... SELECT

```
$sourceQb = $phuriaSQL->createInsert();

$sourceQb->from('transactions', 't');
$sourceQb->addSelect('t.user_id', 'SUM(t.amount)');
$sourceQb->addGroupBy('t.user_id');

$targetQb = $phuriaSQL->createInsertSelect();
$targetQb->into('user_summary', ['user_id', 'total_price']);
$targetQb->selectInsert($sourceQb);

echo $targetQb->buildSQL();
```

```
INSERT INTO user_summary (user_id, total_price)
SELECT t.user_id, SUM(t.amount) FROM transactions AS t GROUP BY t.user_id
```

2.6 Simple UPDATE

```
$qb = $phuriaSQL->createUpdate();

$rootTable = $qb->update('user', 'u');
$qb->addSet("u.updated_at = NOW()");
$qb->andWhere("u.id = 1");

echo $qb->buildSQL();
```

```
UPDATE user AS u SET u.updated_at = NOW() WHERE u.id = 1
```

2.7 Advanced UPDATE

```php
$sourceQb = $phuriaSQL->createSelect();
$sourceQb->addSelect('i.transactor_id');
$sourceQb->addSelect('SUM(i.gross) AS gross');
$sourceQb->addSelect('SUM(i.net) AS net');
$sourceQb->from('invoice', 'i');
$sourceQb->addGroupBy('i.transactor_id');

$qb = $phuriaSQL->update();

$qb->update('transactor_summary', 'summary');
$qb->innerJoin($sourceQb, 'source', 'summary.transactor_id = source.transactor_id');
$qb->addSet('summary.invoiced_gross = source.gross');
$qb->addSet('summary.invoiced_net = source.net');

echo $qb->buildSQL();
```

```sql
UPDATE transactor_summary AS summary INNER JOIN (...) AS source
SET summary.invoiced_gross = source.gross, summary.invoiced_net = source.net
```

```php
$qb = $phuriaSQL->createUpdate();

$qb->update('players', 'p');
$qb->addSet('p.qualified = 1');
$qb->andWhere('p.league = 20');
$qb->addOrderBy('p.major_points DESC, p.minor_points DESC');
$qb->addLimit(20);

echo $qb->buildSQL();
```

```sql
UPDATE players AS p SET p.qualified = 1 WHERE p.league = 20
ORDER BY p.major_points DESC, p.minor_points DESC LIMIT 20
```

CHAPTER 3

Object References

3.1 Table Reference

Methods adding tables (such as *leftJoin*, *from*, *into*) return *TableInterface AbstractTable* instance. Use *AbstractTable* like string will convert this object to reference. All references will be converted to table name (or alias). It allows you to easily change aliases.

```php
$qb = $qbFactory->createSelect();

$userTable = $qb->from('user');
$qb->select("{$userTable}.*");

// Without alias
echo $qb->buildSQL();

$userTable->setAlias('u');

// With alias
echo $qb->buildSQL();
```

```sql
# Without alias
SELECT user.* FROM user;

# With alias
SELECT u.* FROM user AS u;
```

3.2 Column Reference

Table reference is the most commonly used in table's column context. Therefore, here is helper method that returns reference directly to column.

```php
$qb = $qbFactory->createSelect();

$userTable = $qb->from('user', 'u');
$qb->addSelect($userTable->column('username'), $userTable->column('password'));

echo $qb->buildSQL();
```

```sql
SELECT u.username, u.password FROM user u
```

CHAPTER 4

Table Class

4.1 Creating table class

The default implementation of *TableInterface* is *UnknownTable*. For mapping table name to class name is responsible *TableRegistry*.

First you need to crete implementation of *TableInterface*. We highly recommend inheriting from *AbstractTable*.

```
use Phuria\SQLBuilder\Table\AbstractTable;

class AccountTable extends AbstractTable
{
    public function getTableName()
    {
        return 'account';
    }

    public function onlyActive()
    {
        $this->getQueryBuilder()->andWhere($this->column('active'));
    }

    public function joinToContact()
    {
        $qb = $this->getQueryBuilder();
        $userTable = $qb->innerJoin('user', 'u');
        $userTable->joinOn("{$userTable}.id = {$this}.user_id");
        $contactTable = $qb->innerJoin('contact', 'c');
        $contactTable->joinOn("{$contactTable}.user_id = {$userTable}.id");

        return $contactTable;
    }

    public function selectOnlyActiveEmails()
    {
        $this->onlyActive();
        $contactTable = $this->joinToContact();
        $this->getQueryBuilder()->addSelect($contactTable->column('email'));

        return $this;
    }
}
```

Then you need to add the table to configuration (see configuration section). Now when you are referring to this table, you get instance of implemented class.

```
$qb = $qbFactory->createSelect();

$qb->addSelect('*');

$accountTable = $qb->from('account');
$accountTable->onlyActive();

echo $qb->buildSQL();
```

```
SELECT * FROM account WHERE acount.active
```

4.1.1 Relative QueryBuilder

In order to receive instance of *RelativeQueryBuilder*, you have to call *AbstractTable::getRelativeBuilder()*.

```
$qb->from('account')->getRelativeBuilder()
    ->addSelect('@.id');

echo $qb->buildSQL();
```

```
SELECT account.id FROM account
```

Thanks to *RelativeQueryBuilder* every directive `@.` will be changed into related table's name.

CHAPTER 5

Sub Query

To use a sub query like table, pass it as argument (instead of the name of the table). You will get in return an instance of *SubQueryTable* that you can use like normal table (eg. you can set alias).

```
$qb = $phuriaSQL->createSelect();
$subQb->addSelect('MAX(pricelist.price) AS price');
$subQb->from('pricelist');
$subQb->addGroupBy('pricelist.owner_id');

$qb = $phuriaSQL->createSelect();
$subTable = $qb->from($subQb, 'src');
$qb->addSelect("AVG({$subTable->column('price')})");

echo $qb->buildSQL();
```

```
SELECT AVG(src.price) FROM (SELECT MAX(pricelist.price) AS price
FROM pricelist GROUP BY pricelist.owner_id) AS src
```

If you want to use sub query in a different context then you must use object to string reference converter.

```
$qb = $phuriaSQL->createSelect();
$subQb->addSelect('DISTINCT user.affiliate_id');
$subQb->form('user');

$qb = $phuriaSQL->createSelect();
$qb->addSelect("10 = ({$qb->objectToString($subQb)})");

echo $qb->buildSQL();
```

```
SELECT 10 IN (SELECT DISTINCT user.affiliate_id FROM user)
```

At the time of building query *ReferenceParser* will be known what to do with it.

CHAPTER 6

SQL Clauses

6.1 JOIN Clause

To create join, use one of the following methods: *join*, *innerJoin*, *leftJoin*, *rightJoin*, *straightJoin* or *crossJoin*.

Join method signature looks like this:

```
join($table, string $alias = null, string $joinOn = null) : TableInterface
```

Argument *$table* can be one of following types:

- table name
- class name
- closure
- object implementing *QueryBuilderInterface*

```
// Table name:
$qb->join('account');

// Class name:
$qb->join(AccountTable::class);

// Closure:
$qb->join(function (AccountTable $accountTable) {

});

// Another QueryBuilder:
$qb->join($anotherQb);
```

Arguments *$alias* and *$joinOn* are optional. You can set them later directly on the object table.

```
$qb->from('user', 'u');
$qb->join('account', 'a', 'u.id = a.user_id');
```

And equivalent code:

```
$userTable = $qb->from('user', 'u');
$accountTable = $qb->join('account');
$accountTable->setAlias('a');
$accountTable->joinOn("{$userTable->column('id')} = {$accountTable->column('user_id')}");
```

6.1.1 OUTER and NATURAL JOIN

To determine join as *OUTER* or *NATURAL* use methods: *AbstractTable::setNaturalJoin()* or *AbstractTable::setOuterJoin()*

```
$userTable = $qb->leftJoin('user', 'u');
$userTable->setNaturalJoin(true);
$userTable->setOuterJoin(true);
```

6.2 WHERE Clause

```
$qb->andWhere('u.active = 1');
$qb->andWhere('u.email IS NOT NULL');
```

```
WHERE u.active = 1 AND u.email IS NOT NULL
```

6.3 GROUP BY Clause

```
$qb->addGroupBy('YEAR(u.created_at) ASC');
$qb->addGroupBy('u.affiliate_id');
```

```
GROUP BY YEAR(u.country_id) ASC, u.affiliate_id
```

6.3.1 GROUP BY ... WITH ROLLUP

For use the *WITH ROLLUP* clause, use *setGroupByWithRollUp(true)*:

```
$qb->addGroupBy('u.country_id');
$qb->addGroupBy('u.male');
$qb->setGroupByWithRollUp(true);
```

```
GROUP BY u.country_id, u.male WITH ROLLUP
```

6.4 HAVING Clause

```
$qb->addSelect('SUM(i.gross) AS gross');
$qb->addSelect('i.transactor_id');
$qb->from('invoice', 'i');
$qb->addGroupBy('i.transactor_id'):
$qb->andHaving('gross > 1000');
```

```
SELECT SUM(i.gross) AS gross, i.transactor_id
FROM invoice AS i GROUP BY i.transactor_id HAVING gross > 1000
```

6.5 ORDER BY Clause

```php
$qb->addOrderBy('u.last_name ASC');
$qb->addOrderBy('u.first_name ASC');
```

```
ORDER BY u.last_name ASC, u.first_name ASC
```

6.6 LIMIT Clause

```php
$qb->setLimit(10);
$qb->setLimit('10, 20');
$qb->setLimit('10 OFFSET 20');
```