Advanced Querying with CakePHP 3

Agenda

A Short Story

Once upon a time there was a framework that worked like...

```php
$this->Post->recursive = 3;
$this->Post->find('all');
```

And there was much rejoice, and then much sadness.

The Philosophy Behind the New ORM

Understanding the ideas that brought us here will help you use effectively the ORM, and enjoy using databases again.

The R in ORM

In 2010 I wrote a CakePHP datasource plugin for using MongoDB.

I thought it was cool.

```
b0ss: Can you please get last workshop attendees' emails?
me: Wait a minute, I need to write a custom Javascript program to get that data
```

It took much longer than a minute.

- Relational databases won't go away anytime soon
- They are extremely efficient
- SQL is declarative language, even non programmers can get good at it
- Allow you to query your data in any angle

No intentions of being more than an ORM

This means:

- Not going to connect to stuff that is not a relational database
- We can focus on getting the most of of relational features
- No incomplete abstractions

Does not mean:

- That CakePHP 3 can't use NoSQL storage.
- That you will spend hours scratching your head trying to wrap an API around a fixed Interface.

Goals we had in mind

Clean layers

Each part of the subsystem should be usable on its own.

```
// OMG I'm not even using the ORM
$connection->newQuery()->select('*')->from('users');
```

```
// Ok, now I am
$table = TableRegistry::get('users');
$table->find();
```

Extensible column types system

Do you really need ENUM? Go for it!

```
Type::map('enum', 'EnumType');
$users->schema()->columnType('role', 'enum');
```

Lazy (almost) all the things.

- No database connections unless necessary.
- No Queries executed unless needed.

```
// Look ma', no database connection have been made yet!
$users->find('all')->where(['username' => 'jose_zap']);
```

```
// Query was executed, nothing in memory yet
$users->find('all')->where(['username' => 'jose_zap'])->all();
```

```
// Only keep one row in memory at a time
$users->find('all')->where(['username' => 'jose_zap'])->bufferResults(false);
```

Should be fun to work with

- Everything can be an expression

```php
$query = $users->find()->select(['id'])->where(['is_active' => true]);

$anotherQuery->from(['stuff' => $query]);

$anotherQuery->innerJoin(['stuff' => $query]);

$anotherQuery->where(['id IN' => $query]);
```

- Queries can be composed

```php
$premium = $users->find('active')->find('premium')->each(function($user) {
    echo $user->name;
});
```

```php
$subscribers = $users->find('active')->find('subscribedToNewsletter');
$recipients = $premium->append($free)->extract('email');
```

The Setup

```
class CountriesTable extends Table {

    public function initialize(array $config) {
        $this->table('countries');

        $this->belongsTo('Capitals', [
            'foreignKey' => 'capital_id',
        ]);
        $this->hasMany('Cities', [
            'foreignKey' => 'country_id',
        ]);
        $this->hasMany('Languages', [
            'foreignKey' => 'country_id',
        ]);
    }
```

Simple Querying

Monarchies with the largest population

```php
public function findBiggestMonarchies(Query $query) {
    return $query
        ->where(['government_form LIKE' => '%Monarchy%'])
        ->order(['population' => 'DESC']);
}
```

```json
{
    "name": "Japan",
    "population": 126714000
},
{
    "name": "Thailand",
    "population": 61399000
},
{
    "name": "United Kingdom",
    "population": 59623400
},
```

Simple Querying

Republics in the world

```
    public function findRepublics(Query $query) {
        return $query
            ->where(['government_form' => 'Republic'])
            ->orWhere(['government_form' => 'Federal Republic']);
    }
```

SQL Functions

Average life expectancy

```
public function findAverageLifeExpectancy(Query $query) {
    return $query->select(['average_exp' => $query->func()->avg('life_expectancy')]);
}
```

```
{
    "average_exp": 66.48604
}
```

Subqueries

```
    public function findWithHighLifeExp(Query $query) {
        $average = $this->find('findAverageLifeExpectancy');
        return $query
            ->where(['life_expectancy >' => $average])
            ->order(['life_expectancy' => 'DESC']);
    }
```

```
    $countries->find('republics')->find('withHighLifeExp');
```

Republics with high life expectancy:

```
{
    "name": "San Marino",
    "life_expectancy": 81.1
},
{
    "name": "Singapore",
    "life_expectancy": 80.1
},
{
    "name": "Iceland",
    "life_expectancy": 79.4
}
```

Working with associations

```
$this->hasOne('OfficialLanguages', [
    'className' => LanguagesTable::class,
    'foreignKey' => 'country_id',
    'conditions' => ['OfficialLanguages.is_official' => 'T']
]);
```

Official Languages

```
public function findWithOfficialLanguage(Query $query) {
    return $query
        ->contain('OfficialLanguages');
}
```

Association strategies

```
public function findWithSpokenLanguages(Query $query, $options = []) {
    if (!empty($options['languageStrategy'])) {
        $this->Languages->strategy($options['languageStrategy']);
    }
    return $query
        ->contain('Languages');
}
```

Change the strategy:

```
$countries->find('withSpokenLanguages', ['languageStrategy' => 'subquery'])
```

And expect this SQL to be used:

```
SELECT * FROM languages AS Languages
 WHERE country_id IN (SELECT id FROM countries AS Countries)
```

Filtering by associations

Cities with a population larger than Denmark

```
public function findWithCitiesBiggerThanDenmark(Query $query) {
    $denmarkPopulation = $this->find()
        ->select(['population'])
        ->where(['id' => 'DNK']);

    return $query
        ->distinct(['Countries.id'])
        ->matching('Cities', function($q) use ($denmarkPopulation) {
            return $q->where(['Cities.population >' => $denmarkPopulation]);
        });
}
```

Raw SQL and Deep Assocs

I want to learn a new language, so I need to go to a city where that language is spoken by at least 25% of the people who live there:

```
    public function findCityProbability(Query $query) {
        return $query
            ->matching('Countries.Cities', function($q) {
                $prob = $q->newExpr(
                    '(Languages.percentage / 100) *' .
                    '(Cities.population / Countries.population)'
                );

                return $q
                    ->select(['probability' => $prob, 'Cities.name'])
                    ->where(function($exp) use ($prob) {
                        return $exp->gte($prob, 0.25);
                    });
            });
    }
```

Post processing

Things to keep in mind

- Custom finders are required to return a Query object
- Returning an array or a single value is not a Query
- Therefore, you cannot return arrays or any other value

The Solution

- Use `formatResults()`
- Use `mapReduce()`
- Use any of the `Collection` class methods after calling `find()`

Grouping by a Property

```
public function findInContinentGroups(Query $query) {
    $query->formatResults(function($results) {
        return $results->groupBy('continent');
    });
    return $query;
}
```

```
"Africa": [
        {
            "name": "Angola"
        },
        {
            "name": "Burundi"
        },
        {
            "name": "Benin"
        },
        {
            "name": "Burkina Faso"
        }
"America": [...
```

Getting Key - Value Lists

```
public function findOfficialLanguageList(Query $query) {
    $query->formatResults(function($results) {
        return $results->combine('name', 'official_language.language');
    });
    return $query->find('withOfficialLanguage');
}
```

```
{
    "Aruba": "Dutch",
    "Afghanistan": "Pashto",
    "Albania": "Albaniana",
    "Andorra": "Catalan",
    "Netherlands Antilles": "Papiamento",
    "United Arab Emirates": "Arabic",
    "Argentina": "Spanish",
    "Armenia": "Armenian",
    ...
```

Multiple Formatters

```
public function findInRegionalGroups(Query $query) {
    $query
        ->formatResults(function($results) {
            return $results->groupBy('continent');
        })
        ->formatResults(function($results) {
            return $results->map(function($continent) {
                return collection($continent)->groupBy('region');
            });
        });
    return $query;
}
```

```
    "North America": {
        "Caribbean": [
            {
                "name": "Aruba"
            },
            {
                "name": "Anguilla"
            },
            {
                "name": "Netherlands Antilles"
            }
        ...
```

Intelligent Counts

```
$countries->find()
    ->select(function($query) {
        return [
            'average_life_expectancy' => $query->func()->avg('life_expectancy'),
            'continent'

    });
    ->group(['continent'])
    ->count(); // 7
```

Produces the following SQL:

```
SELECT COUNT(*) AS `count`
FROM (
    SELECT (AVG(life_expectancy)), Countries.continent
    FROM countries AS Countries GROUP BY continent
)
AS count_source
```

Pagination: piece of cake!

I have 99 problems...

Custom counting ain't one

- Don't care about actual results counting in a pagination query?
- Prefer using estimates or a different logic?
- Use custom counters!

```
$query = $youtubeVideos->find('superComplexStuff')->counter(function() {
    return Cache::read('estimated_results');
});
```

```
$query->count(); // 10000000
```

There's Plenty More!

But unfortunately, little time...

- Result streaming
- Query caching
- Finder callbacks
- Composite Primary Key searches
- Methods for finding in Tree structures

Thanks for your time

Questions?

https://github.com/lorenzo/cakephp3-examples