

# The Hidden Gems in CakePHP 3

# DebugKit

# Ajax history

## History

10 previous requests available

- Back to current request

1/11/15, 2:02 AM	GET 404 text/html /bookmarks/derps
1/11/15, 2:01 AM	GET 200 text/html /bookmarks
1/11/15, 2:01 AM	GET 200 text/html /users
1/11/15, 2:00 AM	GET 200 text/html /bookmarks/
1/11/15, 2:00 AM	GET 200 text/html /bookmarks/
1/11/15, 2:00 AM	GET 200 text/html /bookmarks/
1/11/15, 2:00 AM	GET 200 text/html /bookmarks/

Cache

Environment

History

Include

Log 6

Request

Session

Sql Log 6 - 3 ms

# I18n and Formatting

# Number::config()

- Intl already knows what is the standard way of representing a number for every locale
- But, you can tweak it a bit.
- You can pre-configure the number formatter per locale

```
Number::config('fr_FR', [  
    'precision' => 3,  
    'places' => 2  
]);  
  
Number::format($someAmount, ['locale' => 'fr_FR']);  
  
$formatter = Number::formatter(['locale' => 'fr_FR']);  
$formatter->format($someAmount);
```

- You can also add more configuration to the formatter object using the INTL constants

# Intl Message Formatter

- You can create feature rich and complex translation strings using the Intl message formatter

```
--('At {1,time} on {1,date}, there was {2} on planet {0,number,integer}.', [  
  $planetNumber,  
  'an exotic dancer',  
  $dateTimeObject  
]);
```

At 13:54 on May, 24 2016, there was an exotic dancer on planet 6543

It supports modifiers such as:

- time
- date
- number
- float
- integer
- currency

# Intl Message Formatter

- You can also select the right plural rules without using the `__n()` function

```
"{gender_of_host, select, "
  "female {"
    "{num_guests, plural, offset:1 "
      "=0 {{host} does not give a party.}"
      "=1 {{host} invites {guest} to her party.}"
      "=2 {{host} invites {guest} and one other person to **her** party.}"
      "other {{host} invites {guest} and # other people to her party.}}}"
    "male {"
      "{num_guests, plural, offset:1 "
        "=0 {{host} does not give a party.}"
        "=1 {{host} invites {guest} to his party.}"
        "=2 {{host} invites {guest} and one other person to **his** party.}"
        "other {{host} invites {guest} and # other people to his party.}}}"
    "other {"
      "{num_guests, plural, offset:1 "
        "=0 {{host} does not give a party.}"
        "=1 {{host} invites {guest} to their party.}"
        "=2 {{host} invites {guest} and one other person to **their** party.}"
        "other {{host} invites {guest} and # other people to their party.}}}"
  }
```

# More context in translations

- Translation messages are more often than not something ambiguous
- This can be solved by using the context functions

```
echo __x('used for uploading a profile image image', 'Upload');  
echo __x('used for uploading a file', 'Upload');
```

- The context is the first argument of the function



# Shell utilities

# Creating a diff between database versions

- Create an initial migration snapshot
- Directly modify your database
- Bake a migration diff
- Profit

```
bin/cake migrations dump
```

```
bin/cake bake migration_diff AddingImportantTrackingColumns
```

# Showing progress in shells

```
$work = function ($progress) {  
    // Do work here.  
    ...  
    $progress->increment(20);  
    $progress->draw();  
  
    // Do more work  
    ...  
    $progress->increment(80);  
    $progress->draw();  
};  
$this->helper('Progress')->output(['callback' => $work]);
```

# Formatting tables in shells

```
$data = [  
  ['Header 1', 'Header', 'Long Header'],  
  ['short', 'Longish thing', 'short'],  
  ['Longer thing', 'short', 'Longest Value'],  
];  
$this->helper('Table')->output($data);
```

# Fancy methods

# Validation builders

## Before

```
$validator->add('some_field', 'rule_name', [  
    'rule' => ['email'],  
    'message' => 'This is wrong'  
]);  
  
$validator->add('another_field', 'rule_name', [  
    'rule' => ['url'],  
    'message' => 'This is wrong too'  
]);
```

## After

```
$validator->email('some_field', 'This is wrong');  
$validator->url('another_field', 'This is wrong too');
```

# So many date methods!!

```
// These getters specifically return integers, ie intval()
var_dump($dt->year);           // int(2012)
var_dump($dt->month);          // int(9)
var_dump($dt->day);            // int(5)
var_dump($dt->hour);           // int(23)
var_dump($dt->minute);         // int(26)
var_dump($dt->second);         // int(11)
var_dump($dt->micro);          // int(123789)
var_dump($dt->dayOfWeek);      // int(3)
var_dump($dt->dayOfYear);      // int(248)
var_dump($dt->weekOfMonth);    // int(1)
var_dump($dt->weekOfYear);     // int(36)
var_dump($dt->daysInMonth);   // int(30)
var_dump($dt->timestamp);      // int(1346901971)
```

- Chronos is a fork of Carbon, adding quite a few improvements on top

# Routes are built with functions

```
$builder = function ($routes) {  
    $routes->addExtensions(['json']);  
    $routes->connect('/', ['action' => 'index']);  
    $routes->connect('/add', ['action' => 'add']);  
    $routes->connect('/:id', ['action' => 'view'], ['pass' => ['id']]);  
    $routes->connect('/:id/edit', ['action' => 'edit'], ['pass' => ['id']]);  
    $routes->connect('/:id/:action/*', [], ['pass' => ['id']]);  
    $routes->connect('/search', ['action' => 'lookup']);  
};  
  
$withDelete = function ($builder) {  
    return function ($routes) use ($builder) {  
        $builder($routes);  
        $routes->connect('/:id/delete', ['action' => 'delete'], ['pass' => ['id']]);  
    }  
};  
  
Router::scope('/posts', ['controller' => 'Posts'], $builder);  
Router::scope('/users', ['controller' => 'Users'], $builder);  
Router::scope('/tags', ['controller' => 'Tags'], $withDelete($builder));
```



# Custom configuration

- Many of the configuration classes can take a function

```
use Monolog\Logger;
use Monolog\Handler\StreamHandler;

Log::config('default', function () {
    $log = new Logger('app');
    $log->pushHandler(new StreamHandler('path/to/your/combined.log'));
    return $log;
});
```

```
ConnectionManager::config('default', function () {
    return new Connection(...);
});
```

- This is handy for testing

# ORM Candy

# Create your own expression classes

```
class MatchAgainstExpression implements ExpressionInterface
{
    public function __construct(array $match, $against, $booleanMode = false)
    {
        $this->fields = $match;
        $this->against = against;
        $this->booleanMode = $booleanMode;
    }

    public function sql(ValueBinder $generator) {
        return sprintf(
            'MATCH (%s) AGAINST (%s) %',
            $this->toSQL($this->fields, $generator),
            $this->toSQL($this->against, $generator),
            $this->booleanMode ? 'IN BOOLEAN MODE' : ''
        );
    }

    ...
}
```

```
$query->select(['score' => new MatchAgainstExpression(['full_name'], $name, true)]);
```

# Global function are OK

## Create functions for custom expressions

```
function matchAgainst(array $fields, $against, $booleanMode = false) {  
    return new MatchAgainstExpression($fields, $against, $booleanMode);  
}
```

```
$query->select(['score' => matchAgainst(['name'], $name, true)]);
```

## Have many of them!

```
$query->where(between('price', 10, 100));
```

# Where did this query come from?

```
$this->Things  
->find('myThings')  
->where($conditions)  
->modifier(sprintf('/* %s - %s */', __FUNCTION__, __LINE__));
```

## Outputs

```
SELECT /* ThingsController::index() - 33 */  
  some_fields  
WHERE  
  conditions
```

## Yes, you can make it generic

- Hint: `Debugger::trace()` in the `buildQuery()` method of the table

# Prevent callbacks in behaviors

You can always add a kill switch flag to each of your custom behaviors:

```
$table->save($entity, ['noAuditLog' => true]); // Incognito mode
```

## In your behavior

```
public function beforeSave(Event $event, Entity $entity, $options)
{
    if (!empty($options['noAuditLog'])) {
        return;
    }
    ...
}
```

# Use the table object in select()

Are you lazy and don't want to list all the table's columns?

```
$table->find()  
  ->select(['only_one_column'])  
  ->contain('Stuff')  
  ...  
  ->select($table->Stuff);
```

It works with instances of Table and Association

# Why \$query->func() ?

You can just this, of course...

```
$beforeQuery->select(['total' => 'SUM(price)'])
```

Why use this instead?

```
$afterQuery->select(['total' => $afterQuery->func()->sum('price')])
```

## The answer is in the result:

```
// Before
{
  "total": "245.2"
}
```

```
// After
{
  "total": 245.2
}
```



# What time is it really?

## Get the right Time object for a column

```
$maybeDateFromInput = $request->data('date_field');  
$type = $table->schema()->columnType('date_column');  
  
$parseDateObject = Type::build($type)->marshall($maybeDateFromInput);
```

This is useful when doing behaviors or plugins where you want to adapt to multiple different database schemas.

# Upsert all the things!

You can load the same CSV structure and do an insert or replace automatically

```
$epilog = collection($columnsn)->map(function($col) {  
    return "$col = VALUES($col)";  
});  
  
$insertQuery = $connection->newQuery();  
$insertQuery =  
    ->insert($columns)  
    ->into('users')  
    ->epilog('ON DUPLICATE KEY UPDATE ' . join(', ', $epilog->toList()));
```

# Chunk that insert

We can now use our insert query

```
collection($csvLines)
  ->chunk(1000)
  ->each(function ($values) use ($insertQuery) {
    $insertQuery
      ->values($values)
      ->execute();
  });
```

# Load more association data

After You have loaded an entity, you can load more associated data later on

```
$post = $this->Posts->get($id);  
  
...  
  
$post = $this->Posts->loadInto($post, ['Comments']);  
count($post->comments);
```

**loadInto() is the secret  
sauce**

Check [jeremyharris/cakephp-lazyload](https://github.com/jeremyharris/cakephp-lazyload)

# Subqueries are a thing

```
$subquery = $comments->find('theBest');
```

## They basically work everywhere

```
$query  
  ->select(['the_alias' => $subquery])  
  
  ->where(['a_column' => $subquery])  
  
  ->from(['table_alias' => $subquery])  
  
  ->innerJoin(['join_alias' => subquery])
```

# Yes, fixtures are classes too

```
class PostsFixture extends TestFixture
{
    public $fields = [
        ...
    ];

    // No $records property

    public function insert($connection)
    {
        $records = json_decode('/path/to/posts_fixtures.json');
        $connection
            ->newQuery()
            ->insert(array_keys($this->fields))
            ->values($records)
            ->execute();
    }
}
```

# Questions?



