## Dancer and DBIx::Class

Stefan Hornburg (Racke) racke@linuxia.de

Dutch Perl Workshop, Utrecht, 25th April 2014

## Dancer and DBIx::Class

Stefan Hornburg (Racke) racke@linuxia.de

Dutch Perl Workshop, Utrecht, 25th April 2014

# Introduction Dancer DBIx::Class Database Administration Dancer::Plugin::DBIC Usage Configuration UTF-8 Create schema dynamically

## Dancer::Session::DBIC

Engines

Example table Serializer

Session expiration

# TableEditor

Installation

Frontend

Routes

Login

Relationships Limitations Configuration

#### Tips and Tricks

Object Inflation

#### Forecast and Contribution

Development

Dancer2

Slides

Dancer is an amazing micro web framework. I started to develop Dancer applications in 2011 after listening to a presentation from sawyer at FOSDEM in Brussels.

There are a couple of reasons that made Dancer my favourite web framework.

# Easy to start with

- Application ready to go
- Syntax easy to understand
- Routes and Keywords

# Easy to expand

- Plugins
- ► Hooks
- Engines

## Solid

- Stable
- Keep behaviour
- Community

## **Applications**

```
Simple Dropbox https:
   //metacpan.org/pod/Dancer::Plugin::Dropbox
```

- .state.gov Websites
  https://eshop.state.gov/
- Monitor for Power Plant

## DBIx::Class

- ▶ ORM
- Objects instead of SQL
- Performance

It took some time to get involved with DBIx::Class for various reasons.

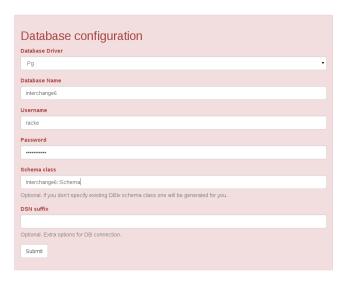
## **Database Administration**

- phpmyadmin
- phppgadmin
- TableEditor

### **TableEditor Features**

- Different database systems MySQL, PostgreSQL, ...
- higher level of abstraction
- modern frontend
- concise source code
- "simple" installation

## Input Database Parameters

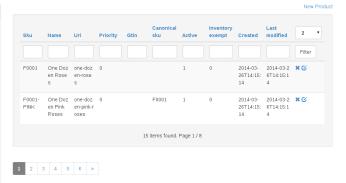


## **View Products**



#### Product - List of items

Tables Attribute Attribute Value Cart Country Group Pricing Media Media Display Media Product Media Type Merchandising Merchandising Navigation Navigation Attribute Navigation Attribute Value Order



## **View Product**

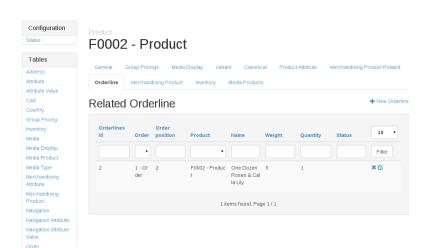
#### Configuration Status Tables Address Attribute Attribute Value Cart Country Group Pricing Media Media Display Media Product Media Type Merchandising Attribute Merchandising Product Navigation Navigation Attribute Navigation Attribute Value Order Orderline Payment Order

# F0002 - Product

General	Orderline C	anonical Produ	ict Attribute Men	chandising Product Related	Merchandising Product
Media Produ	cts Inventory	Group Pricings	Media Display	Variant	
Sku					
F0002					
Name					
One Dozen Ros	es & Calla Lily				
Short description	1				
What says I love	you better than 1	dozen fresh roses v	vith calla lily?		
Description					
Surprise the one	who makes you :	smile, or express you	urself perfectly with th	is stunning bouquet of one do	ozen fresh red roses. This elegan
Price					
49.95					
Uri					
one-dozen-red-i	oses-calla-lilly				
Weight					

## Relationship Orderline

Orderline Payment Order



# Overview Dancer::Plugin::DBIC

- Usage
- Configuration
- ► UTF-8
- Create schema dynamically

## DBIx::Class without Dancer Plugin

```
use Interchange6::Schema;

$schema = Interchange6::Schema->connect(...);

$schema->resultset('User')->search({...});
```

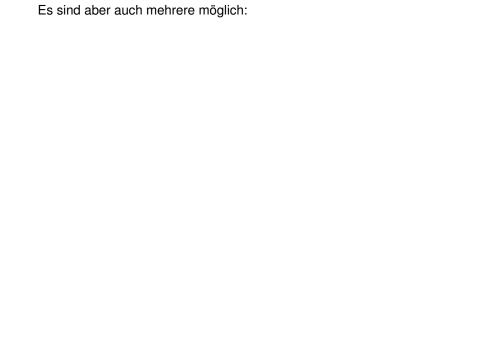
## DBIx::Class with Dancer Plugin

```
use Dancer:: Plugin :: DBIC;
schema->resultset('User')->search({..});
resultset('User')->search({..});
rset('User')->search({..});
```

Im Normalfall verwendet man nur ein Schema in seiner Dancer-Anwendung:

## Configuration

```
plugins:
   DBIC:
    default:
        dsn: dbi:mysql:interchange6
        user: racke
        pass: nevairbe
        schema_class: Interchange6::Schema
```



## Multiple Schemas

```
plugins:
  DBIC:
    default:
      dsn: dbi:mysql:interchange6
      user: racke
      pass: nevairbe
      schema class: Interchange6::Schema
    legacy:
      dsn: dbi:mysql:interchange5
      user: racke
      pass: nevairbe
      schema class: Interchange5::Schema
```



## Multiple Schemas

```
use Dancer::Plugin::DBIC;
schema('legacy')->resultset('UserDb')->search({..});
```

Im Gegensatz zu Dancer::Plugin::Database bietet das DBIC-Plugin keine automatische Unterstützung für UTF-8. Also ist die entsprechende DBI-Option in der Konfiguration einzutragen, hier für

MySQL:

## UTF-8 for MySQL

```
plugins:
    DBIC:
    default:
    dsn: dbi:mysql:interchange6
    user: racke
    pass: nevairbe
    schema_class: Interchange6::Schema
    options:
        mysql_enable_utf8: 1
```

Die Optionen für die gängigen Datenbanken in der Übersicht:

SQLite sqlite\_unicode: 1

MySQL mysql\_enable\_utf8: 1

PostgreSQL pg\_enable\_utf8: 1

praktisch für den TableEditor.

Das DBIC-Plugin erzeugt dynamisch ein DBIx::Class::Schema, wenn die Schema-Klasse (schema\_class) nicht angegeben wird. Dazu ist das Modul DBIx::Class::Schema::Loader erforderlich. Dies ist nicht empfehlenswert für den Produktionseinsatz, jedoch

# Create schema dynamically

- schema\_class missing in configuration
- DBIx::Class::Schema::Loader
- test and development
- TableEditor

# **Engines**

- Templates TT, Xslate, Flute, ...
- SessionsStorable, Database, DBIC
- LoggerFile, Syslog
- Serializer JSON, YAML, XML

Die Sessionengines werden in Dancer für gewöhnlich transparent für den Anwendungscode in der Konfiguration eingerichtet:

# Configuration

session name of session engine (DBIC) session\_options options session\_expires expiration date

Das ermöglicht es, auf dem Liveserver eine effizientere Engine zu verwenden (z.B. Storable) und auf dem Entwicklungsserver eine Engine, die einem beim debuggen hilft (z.B. YAML). Die Optionen für Dancer::Session::DBIC ähneln der Konfiguration von Dancer::Plugin::DBIC, zusätzlich können wir festlegen wie die Sessions aus der Datenbank abgerufen werden können:

resultset DBIx::Class resultset

id column primary key

data column field for session data

Das sieht dann z.B. für Interchange6::Schema (Version 0.015) so aus:

## Configuration

```
session: "DBIC"
session options:
 dsn: dbi:mysql:interchange6
 user: racke
 pass: nevairbe
 schema class: Interchange6::Schema
  resultset: Session
 id column: sessions_id
 data column: session data
session expires: 12 hours
```

Die Konfiguration kann aber ebenso im Hauptmodul stattfinden:

## Configuration

```
set session => 'DBIC';
set session_options => {schema => schema};
```

Folgendermaßen sieht die Tabelle sessions aus, die vom Schema Interchange6::Schema (Version 0.015) erzeugt wird:

# Example table

```
CREATE TABLE 'sessions' (
    'sessions_id' varchar(255) NOT NULL,
    'session_data' text NOT NULL,
    'created' datetime NOT NULL,
    'last_modified' datetime NOT NULL,
    PRIMARY KEY ('sessions_id')
) ENGINE=InnoDB;
```

## Serializer

```
set 'session_options' => {
    schema => schema,
    serializer => sub { YAML::Dump(@_); },
    deserializer => sub { YAML::Load(@_); },
};
```

Beim Überschreiten der erlaubten Ablaufzeit wird die Sitzung ungültig, sie wird jedoch nicht in der Datenbank gelöscht. Dafür ist ein Skript zur regelmäßigen Löschung der abgelaufenen Datensätze erforderlich.

JSON andere DBIC connection? tests?

# Session expiration

- remove old sessions from database
- ▶ Interchange6::Schema::Resultset::Session

```
$schema->resultset('Session')->expire('12 hours');
```

Im günstigsten Fall kann die Installation mit 4 Schritten erledigt

werden:

#### Installation

```
git clone https://github.com/interchange/TableEditor
cd TableEditor
cpanm .
./bin/app.pl
```

## Driver

- DBD::mysql
- ▶ DBD::Pg
- **.**..

Das Frontend für den TableEditor ist mit Angular und Bootstrap erstellt. Das Theme kann sehr einfach durch Austausch der CSS-Datei für

Bootstrap geändert werden.

#### Routes

```
get '/:class/:id' => require_login sub {
    # retrieve database record and add relationships
    ...
    return to_json($data, {allow_unknown => 1});
};
```

Für die Integration von Authentifizierung in eine Dancer-Anwendung

empfehlen wir wärmestens das Auth::Extensible Plugin.

# Login

- Dancer::Plugin::Auth::Extensible
- Provider
  - Unix
  - DBIC
- Database (planned)

Beziehungen werden automatisch angezeigt.

# Relationships

- might\_have
- has\_many
- ▶ belongs\_to
- many\_to\_many

# Filter Es fehlen Felder in related orderline (Übersicht) Different DBIC keys

## Paging

## Limitations

- Primary key for one column only
- Speed (complex schemas)
- Error handling

# Configuration

- Auth::Extensible
- DBIC
  - ▶ default

## Planned Features

- ► Search (Solr)
- Select schema
- Debian packages

#### User role

## User role

```
$role = $schema->resultset('Role')->find(3);
# primary key
$role->id;
# name and label
$role->name;
$role->label;
```

The search functions of DBIx::Class return objects, but in some situations you prefer a hashref.

# Object vs Hashref

- Debug / Logs
- ► Templates
- ► API / JSON
- Speed

Das Git-Repository für den TableEditor befindet sich auf Github:

# Development

https://github.com/interchange/TableEditor

Was ist mit Dancer2?

Für Dancer2 existiert bereits ein Plugin:

https://metacpan.org/pod/Dancer2::Plugin::DBIC Die Sessionengine und der TableEditor wurden noch nicht auf Dancer2 portiert.

#### Dancer2

#### Slides

Slides: http://www.linuxia.de/talks/nlpw2014/dancer-dbic-en-beamer.pdf