

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

Database configuration

Database Driver

Pg

Database Name

interchange6

Username

racke

Password

Schema class

Interchange6::Schema

Optional. If you don't specify existing DBIx schema class one will be generated for you.

DSN suffix

Optional. Extra options for DB connection.

Submit

View Products

Configuration

Status

Tables

Address

Attribute

Attribute Value

Cart

Country

Group Pricing

Inventory

Media

Media Display

Media Product

Media Type

Merchandising

Attribute

Merchandising

Product

Navigation

Navigation Attribute

Navigation Attribute

Value

Order

- ...

Product - List of items

[New Product](#)

Sku	Name	Uri	Priority	Gtin	Canonical sku	Active	Inventory exempt	Created	Last modified	2 ▾
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	Filter
F0001	One Dozen Roses	one-doz-en-rose-s	0			1	0	2014-03-26T14:15:14	2014-03-26T14:15:14	✕ 🔗
F0001-PINK	One Dozen Pink Roses	one-doz-en-pink-roses	0		F0001	1	0	2014-03-26T14:15:14	2014-03-26T14:15:14	✕ 🔗

15 items found. Page 1 / 8

1 2 3 4 5 6 »

View Product

Configuration

Status

Tables

Address

Attribute

Attribute Value

Cart

Country

Group Pricing

Inventory

Media

Media Display

Media Product

Media Type

Merchandising

Attribute

Merchandising

Product

Navigation

Navigation Attribute

Navigation Attribute

Value

Order

Orderline

Payment Order

Product

F0002 - Product

General

Orderline

Canonical

Product Attribute

Merchandising Product Related

Merchandising Product

Media Products

Inventory

Group Pricing

Media Display

Variant

SKU

F0002

Name

One Dozen Roses & Calla Lily

Short description

What says I love you better than 1 dozen fresh roses with calla lily?

Description

Surprise the one who makes you smile, or express yourself perfectly with this stunning bouquet of one dozen fresh red roses. This elegant

Price

49.95

URI

one-dozen-red-roses-calla-lilly

Weight

Relationship Orderline

Configuration

Status

Tables

Address

Attribute

Attribute Value

Cart

Country

Group Pricing

Inventory

Media

Media Display

Media Product

Media Type

Merchandising

Attribute

Merchandising

Product

Navigation

Navigation Attribute

Navigation Attribute

Value

Order

Orderline

Payment Order

Product

F0002 - Product

General

Group Pricings

Media Display

Variant

Canonical

Product Attribute

Merchandising Product Related

Orderline

Merchandising Product

Inventory

Media Products

Related Orderline

[+ New Orderline](#)

Orderlines id	Order	Order position	Product	Name	Weight	Quantity	Status	10 ▾
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	Filter
2	1 - Or der	2	F0002 - Produc t	One Dozen Roses & Cal la Lily	5	1		✕ 🔍

1 items found. Page 1 / 1

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
```

Es sind aber auch mehrere möglich:

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

Das Schema `legacy` wird dann wie folgt verwendet:

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`

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 praktisch für den TableEditor.

Create schema dynamically

- ▶ `schema_class` missing in configuration
- ▶ `DBIx::Class::Schema::Loader`
- ▶ test and development
- ▶ `TableEditor`

Engines

- ▶ Templates
TT, Xslate, Flute, ...
- ▶ Sessions
Storable, Database, DBIC
- ▶ Logger
File, 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 wärmstens 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

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

interchange6=> **select** * from roles where roles_id = 3;

roles_id	name	label
----------	------	-------

3	user	User
---	------	------

(1 row)

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

Plugin::DBIC <https://metacpan.org/pod/Dancer2::Plugin::DBIC>

Session::DBIC <https://metacpan.org/pod/Dancer2::Session::DBIC> ported 24th April

TableEditor not yet ported

Slides

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