# DBIx::Class effektiv nutzen

Stefan Hornburg (Racke)

German Perl Workshop 2016

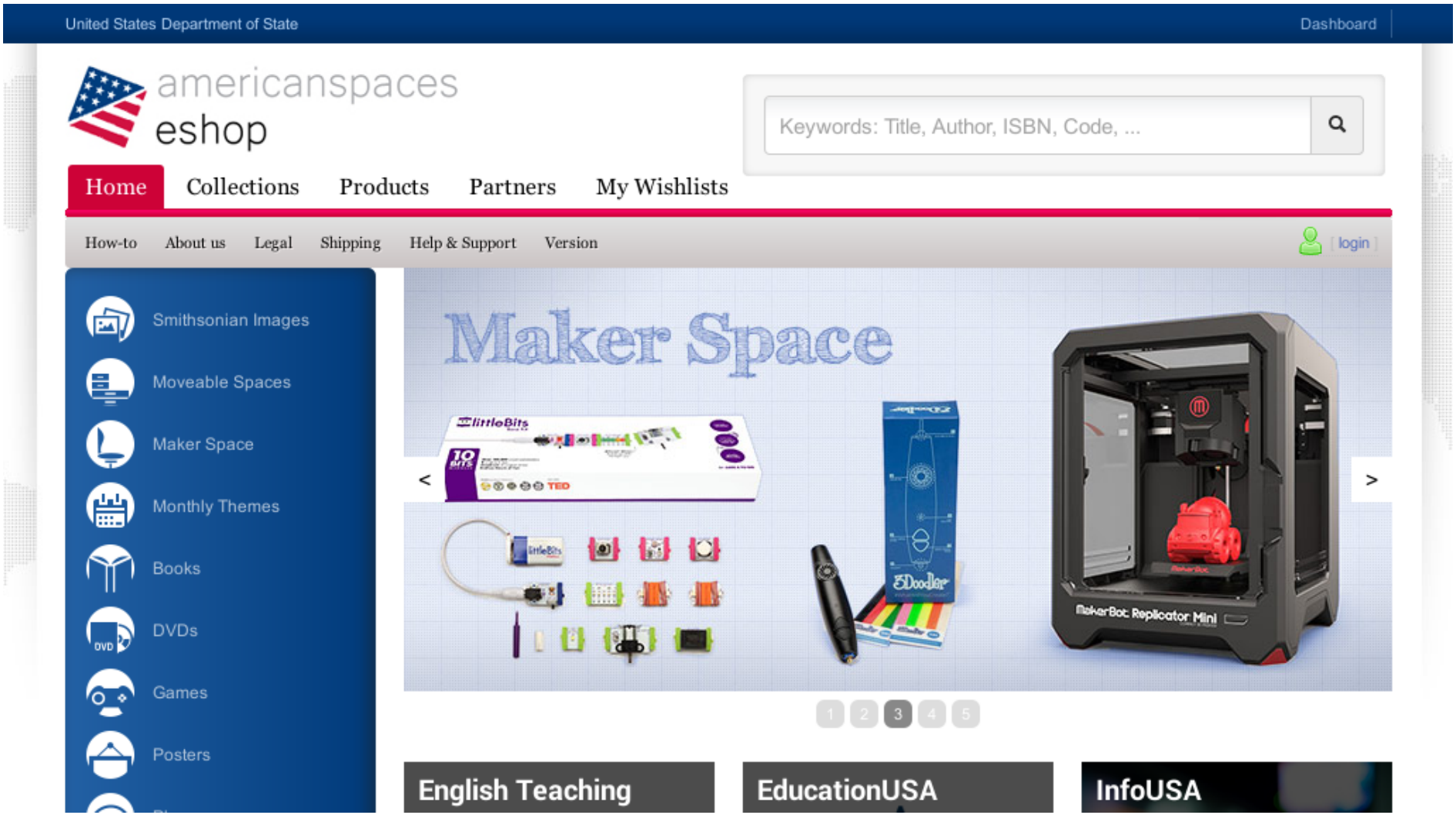
# Can of Worms

# Best Thing Since Sliced Bread

# /me

- Nick: racke
- Selbstständiger Programmierer seit 1998
- Webanwendungen
- Ecommerce
- Datendanken
- Kunden in Österreich, Schweiz, USA, ...

# eShop


United States Department of State
Dashboard
americanspaces
eshop
Keywords: Title, Author, ISBN, Code, ...
Home
Collections
Products
Partners
My Wishlists
How-to
About us
Legal
Shipping
Help & Support
Version
login
Smithsonian Images
Moveable Spaces
Maker Space
Monthly Themes
Books
DVDs
Games
Posters
Maker Space
littleBits
3Doodler
MakerBot Replicator Mini
English Teaching
EducationUSA
InfoUSA

# Interchange

# Interchange

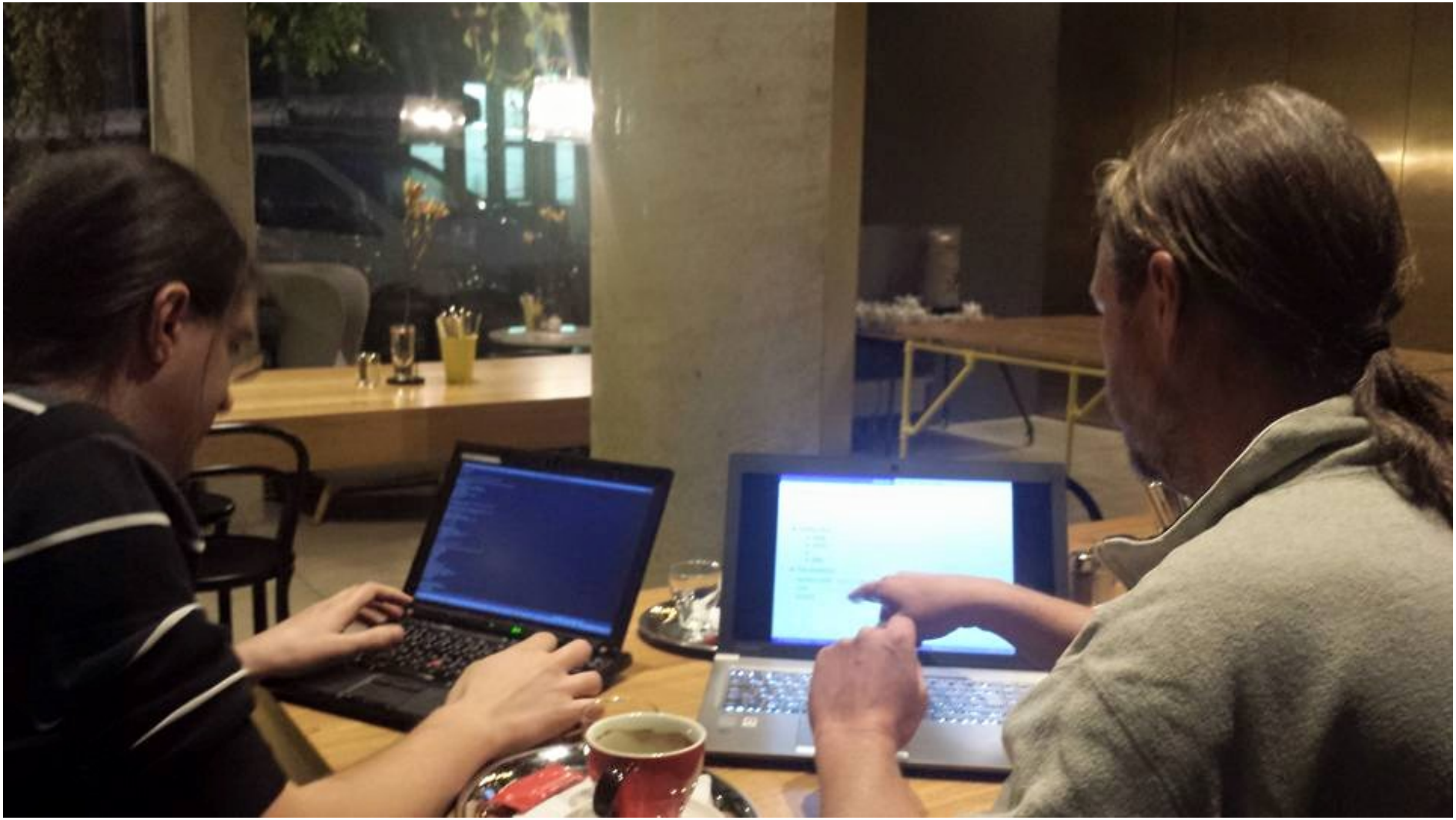
- Embedded SQL
- ~~Modern Perl~~

# Dancer / DBIx::Class

- Dancer
- DBIx::Class
- Interchange6

# Perl Dancer Conference

# DBIx::Class Training



# Konferenz-Website

- ACT 2014
  - Legacy code
  - Keine Kontrolle über den Server
- www.perl.dance 2015
  - Dancer / DBIx::Class
  - Features aus ACT
  - neue Features
  - `https://github.com/interchange/Perl-Dancer-Conference`

# Best Thing Since Sliced Bread

# Best Thing Since Sliced Bread

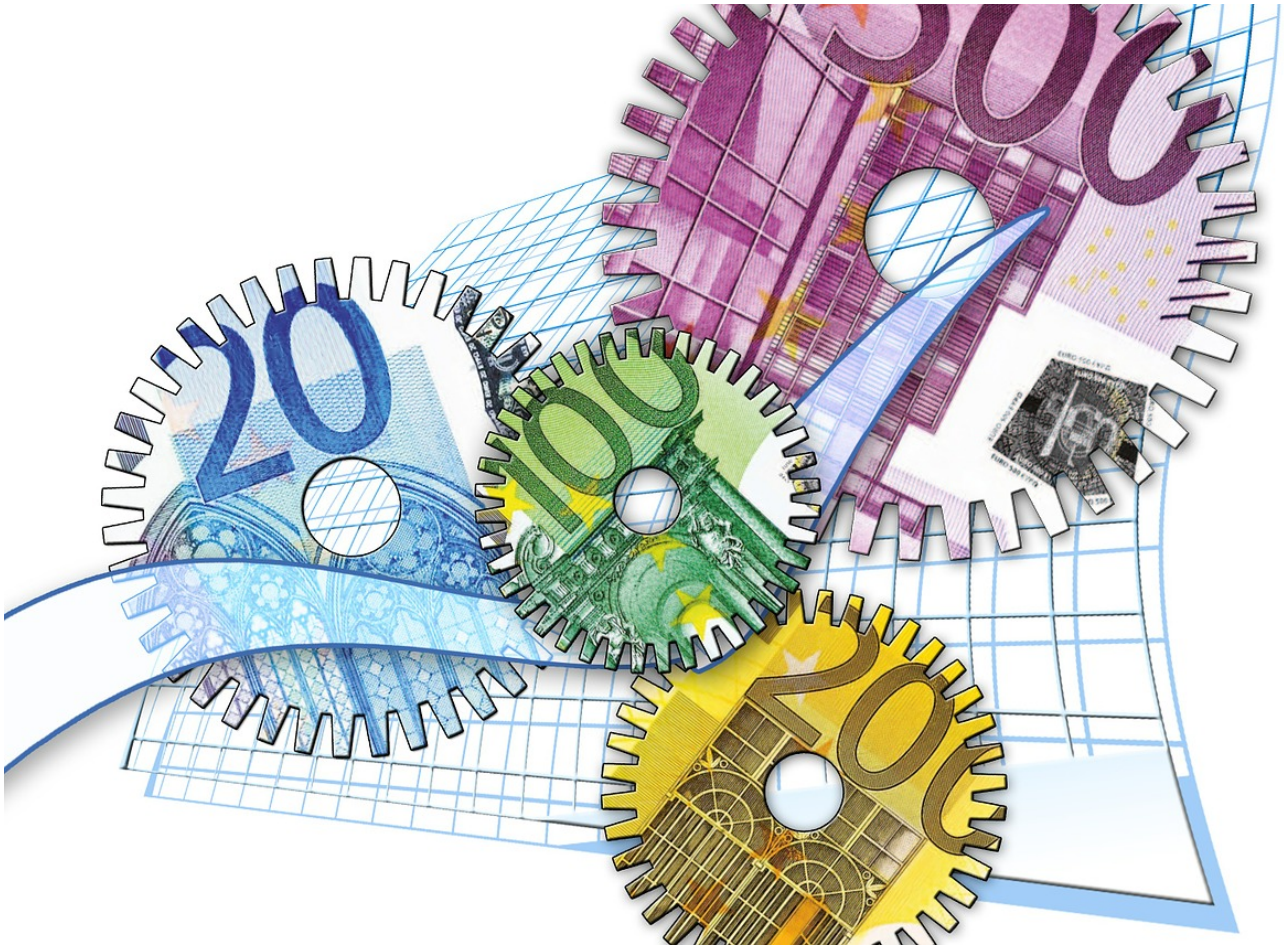
- OO anstatt von SQL
- Abstrahierung von SQL-Implementierungen
- Businesslogik
- Performance
- “ResultSet” Features
- Ökosystem
  - IRC / Mailing list
  - Komponenten
  - Helpers

# Can of Worms

# Can of Worms

- SQL::Abstract
- Klassennamen (Result, ResultSet)
- Basiert auf C3 anstatt Moo(se)
- Dokumentation beschränkt auf DBIx::Class

# Businesslogik



# Vorteile Businesslogik

- Mehrere Verbraucher
  - Webanwendungen
  - Cronjobs, Skripte
  - Testumgebungen
- Implementation verbergen
  - SQL-Dialekte
  - Anwendungsansicht
- Änderungen / DRY

# Beispiele Businesslogik

- Preisberechnungen
  - Angebote
  - Rabatt
- Synchronisierung ERP

# Performance

- Erfahrung
- Testsuiten
- Use cases

# Hotline

# Literal SQL

```
$schema->resultset('PlaceVisited')->search(
{
  users_id => {
    -in => \[
      'SELECT users_id FROM users WHERE users_id > ?',
      2
    ]
  }
});
```

# SQL Kopfschmerzen

# DBIx::Class Mengenlehre

- ORM
  - Objekte statt Array / Hashes
  - Abstrahierung SQL-Implementierung
  - ...
- ResultSet
  - Verkettung
  - Relationship Traversal
  - Subqueries
  - ...

# Using ResultSet

- Simple SQL query
- Turn into DBIx::Class search
- Use ResultSet features

# Simple SQL query

List of talks on last day of Perl Dancer Conference:

```
SELECT talks_id, author_id, conferences_id, duration,
title, tags, abstract, url, comments, accepted, confirmed,
lightning, scheduled, start_time, room, survey_id
FROM talks
WHERE accepted is TRUE
AND conferences_id = 1
AND room != ''
AND start_time >= '2015-10-22 00:00:00'
AND start_time <= '2015-10-23 00:00:00'
```

# Simple SQL query

With DBIx::Class:

```
my $talks = $schema->resultset('Talk')->search(
    {
        -bool           => 'accepted',
        conferences_id => 1,
        room            => { '!=' => '' },
        start_time      => {
            '>=' => '2015-10-22 00:00:00'
            '<=' => '2015-10-23 00:00:00'
            },
    },
);
```

# Wahrheit über das ResultSet


WRONG
WAY
HARDING

# Wahrheit über das ResultSet

~~ResultSet~~

```
isa(Abfrageplan);
```

# Wahrheit über das ResultSet

Hier wird keine SQL-Abfrage ausgeführt:

```
my $talks = $schema->resultset('Talk')->search(...);
```

Hier schon:

```
my $first_talk = $schema->resultset('Talk')
                 ->search(...)->first;
```

# ResultSet-Baukasten

# ResultSet-Baukasten

- Verkettung
- Vordefinierte Suche
- Spezifische JOIN-Bedingungen
- ...

# Suche ohne Verkettung

```
$schema->resultset('Product')->search({
    active => 1,
    canonical_sku => undef,
});
```

# Suche mit Verkettung

```
$schema->resultset('Product')->search({
    active => 1,
})->search({
    canonical_sku => undef,
});
```

# Verkettung mit Update

```
$schema->resultset('Product')->search({
    manufacturer => 'Out of Fashion',
})->update({
    active => 0,
});
```

# Vordefinierte Suche

```
package Interchange6::Schema::ResultSet::Product;

sub active {
    my $self = shift;

    return $self->search({ $self->me('active') => 1 });
}

sub canonical_only {
    my $self = shift;

    return $self->search({
       $self->me('canonical_sku') => undef });
}
```

# Vordefinierte Suche anwenden

```
$schema->resultset('Product')->active->canonical_only;
```

# Vordefinierte Suche: Produktvarianten

```
sub with_variant_count {
 my $self = shift;
 return $self->search(
  undef,
  {
   '+columns' => {
     variant_count =>
      $self->correlate('variants')->count_rs->as_query
   }
  }
 );
}
```

# Spezifische JOIN-Bedingungen

```
__PACKAGE__->has_many(
   "addresses",
   "CalevoMobile::Schema::Result::Address",
   { "foreign.username" => "self.username" },
   { cascade_copy => 0, cascade_delete => 0 },
);
```

# Spezifische JOIN-Bedingungen

```
__PACKAGE__->has_many(
    "shipping_addresses",
    "CalevoMobile::Schema::Result::Address",
    sub {
        my $args = shift;

        return {
            "$args->{foreign_alias}.username" =>
                { -ident => "$args->{self_alias}.username",
            "$args->{foreign_alias}.type" => 'shipping',
            "$args->{foreign_alias}.archived" => 0,
        }
    },
    { cascade_copy => 0, cascade_delete => 0 },
);
```

# Erweiterungen

- Candy
- Schema "vererben"
- Zusatzattribute
- Komponenten
- Helpers
- Deployment Handler

# Zucker für DBIx::Class

# Vanilla Result Class

```
package TravelDance::Schema::Result::Country;
use warnings;
use strict;
use base 'DBIx::Class::Core';

__PACKAGE__->table('countries');

__PACKAGE__->add_columns(
    country_iso_code => {
        data_type => "char",
        size      => 2,
    },
    name => {
        data_type => "varchar",
        size      => 255,
    },
);
```

# Candy Result Class

```
package TravelDance::Schema::Result::Country;
use TravelDance::Schema::Candy;

primary_column country_iso_code => {
    data_type => "char",
    size      => 2
};

column name => {
    data_type => "varchar",
    size      => 255
};
```

# Schemas vererben: Anwendungsbeispiele

- Generisches Schema, z.B. `Interchange6::Schema`
- Anwendung mit ähnlichen Datenbanken

# Schemas vererben

- Tabellen hinzufügen
- Spalten hinzufügen
- Beziehungen hinzufügen

# Beispiel Vererbung

```
package PerlDance::Schema;
our $VERSION = 16;

use Interchange6::Schema::Result::User;
package Interchange6::Schema::Result::User;

__PACKAGE__->add_columns(
    bio => { data_type => "varchar", size => 2048,
             default_value => '' },
    media_id =>
      { data_type => "integer", is_nullable => 1 },
    pause_id => { data_type => "varchar", size => 128,
        default_value => '' },
    t_shirt_size => { data_type => "varchar", size => 8,
      is_nullable => 1 },
);
```

# Beispiel Vererbung

```
package PerlDance::Schema;

use base 'Interchange6::Schema';

Interchange6::Schema->load_namespaces(
    default_resultset_class => 'ResultSet',
    result_namespace        =>
        [ 'Result', '+PerlDance::Schema::Result' ],
    resultset_namespace     =>
        [ 'ResultSet', '+PerlDance::Schema::ResultSet' ],
);
```

# Problemstellung

- Konfiguration
- angemeldeter Benutzer

# Schemainstanz erzeugen

```
use Interchange6::Schema;

my $schema = Interchange6::Schema->connect(
  'dbi:Pg:dbname=perldance', 'racke', 'nevairbe',
);
```

# Zusatzattribute

```
__PACKAGE__->mk_group_ro_accessors(
    inherited => (
        [ 'current_user' => '_ic6_current_user' ]
    )
);

__PACKAGE__->mk_group_wo_accessors(
    inherited => (
        [ 'set_current_user' => '_ic6_current_user' ]
    )
);
```

# Zusatzattribute

```
package Dancer::Plugin::Interchange6;

hook before => sub {
    shop_schema->set_current_user(
        logged_in_user || undef
    );
};
```

# Tree::Adjacency Komponente

```
package Interchange6::Schema::Result::Navigation;

...

__PACKAGE__->load_components(qw( Tree::AdjacencyList ... ))

...

__PACKAGE__->add_columns(
    ...
  "parent_id",
   { data_type => "integer", default_value => 0, is_nullable
    ...
);

...
```

# Tree::Adjacency Methoden

- ancestors
- children
- siblings

# DBIx::Class Helpers

Typische Anwendungsfälle für DBIx::Class vereinfachen.

# DBIx::Class Helpers

Arthur Axel "fREW" Schmidt

# DBIx::Class Helpers

- Helper::Schema::QuoteNames
- Helper::ResultSet::Me
- Helper::Row::ProxyResultSetMethod
- Helper::Row::OnColumnChange

# Helper::QuoteNames

- Maskierung von reservierten Wörtern
- Änderungen zwischen Engines und Versionen
- e.g. MySQL
  - `select user from userdb` => crash
  - `select ‘user‘ from ‘userdb‘` => works
- `quote_names` in connection info

# Helper::QuoteNames

```
package Interchange6::Schema;

use base 'DBIx::Class::Schema';

__PACKAGE__->load_components(
    'Helper::Schema::QuoteNames'
);

...
```

# Helper::Resultset::Me

```
sub active {
    my $self = shift;

    return $self->search({
        $self->current_source_alias . ".active" => 1,
    });
}
```

# Helper::Resultset::Me

```
sub active {
    my $self = shift;

    return $self->search({
        $self->me('active') => 1,
    });
}
```

# Shortcuts

- columns
- like
- hri

# columns Shortcut

`columns` shortcut:

```
$rs->columns([qw/ first_name last_name /]);
```

gleichbedeutend mit:

```
$rs->search( undef, {
    columns => [qw/ first_name last_name /]
} );
```

# like Shortcut

`like` shortcut:

```
$rs->like( 'city', 'region', '%York' );
```

oder:

```
$rs->like([ 'city', 'region' ], '%York' );
```

gleichbedeutend mit:

```
$rs->search(
    { city => { -like => '%York' },
    { region => { -like => '%York' },
);
```

# HashRefInflator

```
my $rs = $schema->resultset('Country')->search({}, {
    result_class
      => 'DBIx::Class::ResultClass::HashRefInflator',
 });
```

# HashRefInflator mit HRI helper

```
my $rs = $schema->resultset('Country')->search({})->hri;

# since 'search' here is redundant we can just use:
my $rs = $schema->resultset('Country')->hri;
```

# Tickets

```
$tokens->{tickets} = [
    $schema->resultset('Conference')
        ->find( $conferences_id )
            ->tickets->active->prefetch('inventory')
                ->hri->all
];
```

# Helper::Row::ProxyResultSetMethod

```
package Interchange6::Schema::Product;

use Interchange6::Schema::Candy -components => [
    qw( Helper::Row::ProxyResultSetMethod );
];

proxy_resultset_method 'variant_count';
```

# Helper::Row::OnColumnChange

Führe Aktionen aus, wenn sich der Wert einer Spalte ändert:

- before_column_change
- around_column_change
- after_column_change

# Helper::Row::OnColumnChange

```
package TravelDance::Schema::Result::User;

use TravelDance::Schema::Candy -components =>
    [qw(Helper::Row::OnColumnChange
        InflateColumn::DateTime)];

use DateTime;

column last_password_change => {
    data_type => timestamp,
};
```

# Helper::Row::OnColumnChange

On password change:

- neues Passwort mit dem alten vergleichen
- Wert in `last_password_change`-Spalte aktualisieren

# Helper::Row::OnColumnChange

```
after_column_change password => {
    method   => 'change_password',
    txn_wrap => 1,  # wrap it all in a transaction
};
```

# Helper::Row::OnColumnChange

```
sub change_password {
    my ( $self, $old_value, $new_value ) = @_;
    if ( $self->check_password($new_value) ) {
        $self->throw_exception("Password not changed")
    }
    else {
        $self->update(
            { last_password_change => DateTime->now() }
        );
    }
}
```

# Andere nützliche Helper

- Helper::Schema::DateTime
- Helper::ResultSet::CorrelateRelationship
- Helper::ResultSet::RemoveColumns
- Helper::ResultSet::Random
- Helper::Row::NumifyGet

# Deployment

- Generierung aus der Datenbank
  - `dbicdump`
- Generierung aus dem Schema
  - `DBIx::Class::DeploymentHandler`

# Generierung aus der Datenbank: dbicdump

```
dbicdump -o dump_directory=/home/dance/TravelDance/lib
         TravelDance::Schema
         dbi:Pg:dbname=perldance
```

# Generierung aus der Datenbank: dbicdump

- `dbicdump` mehrfach anwenden
- zusätzlichen Methoden

# Nachteile dbicdump

- Komponenten, Helpers
- Candy
- generische Schemas

# Deployment Handler

- DBIx::Class::DeploymentHandler
- Datenbankänderungen einspielen
- Datenbank upgraden
- Datenbank downgraden

# Skripte für Deployment Handler

dh-prepare-version-storage Installation Datenbanktabelle vorbereiten

dh-install-version-storage Installation Datenbanktabelle einspielen

dh-prepare-upgrade Upgrade vorbereiten

dh-upgrade Upgrade einspielen

# Vorgehensweise für Updates

- Backup anlegen
- Schema ändern
- Skripts hinzufügen
- Versionsnummer erhöhen
- Upgrade vorbereiten
- SQL-Befehle kontrollieren
- Upgrade einspielen

# Schema ändern

- Tabelle hinzufügen
- Spalte hinzufügen
- Spalte ändern

# Versionsnummer erhöhen

- Natürliche Zahlen: 1, 2, 3, ...
- Erhöhen: 1 => 2

# Version 1

```
package TravelDance::Schema;
use warnings;
use strict;
use base 'DBIx::Class::Schema';

our $VERSION = 1;

__PACKAGE__->load_namespaces();

1;
```

# Version 2

```
package TravelDance::Schema;
use warnings;
use strict;
use base 'DBIx::Class::Schema';

our $VERSION = 2;

__PACKAGE__->load_namespaces();

1;
```

# Upgrade vorbereiten

```
my $dh      = DBIx::Class::DeploymentHandler->new(
    {
        schema                  => $schema,
        databases               => 'MySQL',
        sql_translator_args => { add_drop_table => 0 }
    }
);
$dh->prepare_deploy;
$dh->prepare_upgrade(
    {
        from_version => $dh->database_version,
        to_version => $dh->schema_version
    }
);
```

# Eigene Skripte hinzufügen

- Deployment
- für alle Upgrades
- bestimmte Upgrades

# Guru update

sql/_common/upgrade/13-14/010-gurus.pl

```
#!perl
sub {
    my $schema = shift;
    $schema->resultset('User')->search(
        {
            username => {
                -in => [
                    'xsawyerx@cpan.org',
                    'russell.jenkins@strategicdata.com.au',
                    'rabbit@rabbit.us',
                ]
            }
        }
    )->update( { guru_level => 100 } );
};
```

# Verzeichnisse und Dateien

sql/PostgreSQL Datenbankspezifische SQL-Skripte

- sql/PostgreSQL/deploy/1/001-auto.sql Deploy Version 1
- sql/PostgreSQL/upgrade/1-2/001-auto.sql Upgrade 1 => 2

sql/_common Eigene Skripte

- sql/_common/upgrade/_any Alle Upgrades
- sql/_common/upgrade/1-2 Upgrade 1 => 2

sql/_deploy Strukturdateien (YAML)

# Deployment Handler CLI

```
#! /usr/bin/env perl

use strict;
use warnings;
use PerlDance::Schema;
use DBIx::Class::DeploymentHandler::CLI;

my $schema = PerlDance::Schema->connect('perldance');

my $dh_cli = DBIx::Class::DeploymentHandler::CLI->new(
    schema => $schema,
    databases => 'PostgreSQL',
    args => \@ARGV,
);

if (my $ret = $dh_cli->run) {
   print $ret, "\n";
}
```

# Abschluß

- Resources
- Slides
- Fragen

# Resources

- Extensive Documentation
  - `DBIx::Class::Manual::*`
  - `DBIx::Class::Manual::ResultClass`
  - `DBIx::Class::ResultSet`
  - `DBIx::Class::Relationship::*`
- Perl Advent Calendar 2012: Set-based DBIx::Class by fRew

# Interchange Resources


- Interchange6::Schema
- Demo Shop
- Perl Dancer Conference

# Slides

Slides: `http://www.linuxia.de/talks/gpw2016/dbic-pr-de-beamer.pdf`

# Fragen

Fragen ?