Geschichten aus der Praxis zu DBIx::Class

Stefan Hornburg (Racke)

Hamburg Perl Mongers, 5. Dezember 2017

/me

- Nick: racke
- Selbstständiger Programmierer seit 1998
- Webanwendungen
- Ecommerce
- Datendanken
- Provisionierung mit Ansible
- Kunden in Deutschland, Österreich, Schweiz, USA, Australien, ...

Einführung

- Architektur
- Vorteile

DBIx::Class Architektur

- Datenbank *perldance*
- Schemaobjekt *PerlDance::Schema*

DBIx::Class Architektur

- Tabellen
  - *countries*
  - *users*
- **ResultSet-Klassen**
  - *PerlDance::Schema::ResultSet::Country*
  - *PerlDance::Schema::ResultSet::User*

DBIx::Class Architektur

- Datensätze

```
{"country_iso_code": "de", "name" : "Germany"}
{"username": "racke", "email" : "racke@racke.pm"}
```

- **Result-Klassen**
  - *PerlDance::Schema::Result::Country*
  - *PerlDance::Schema::Result::User*

DBIx::Class Architektur

- SQL-Befehle
  - INSERT
  - UPDATE
  - DELETE
  - SELECT

DBIx::Class Architektur

INSERT

```
$country = $schema->resultset('Country')->create( { ... } );
```

UPDATE

```
$country->update( { ... } );
```

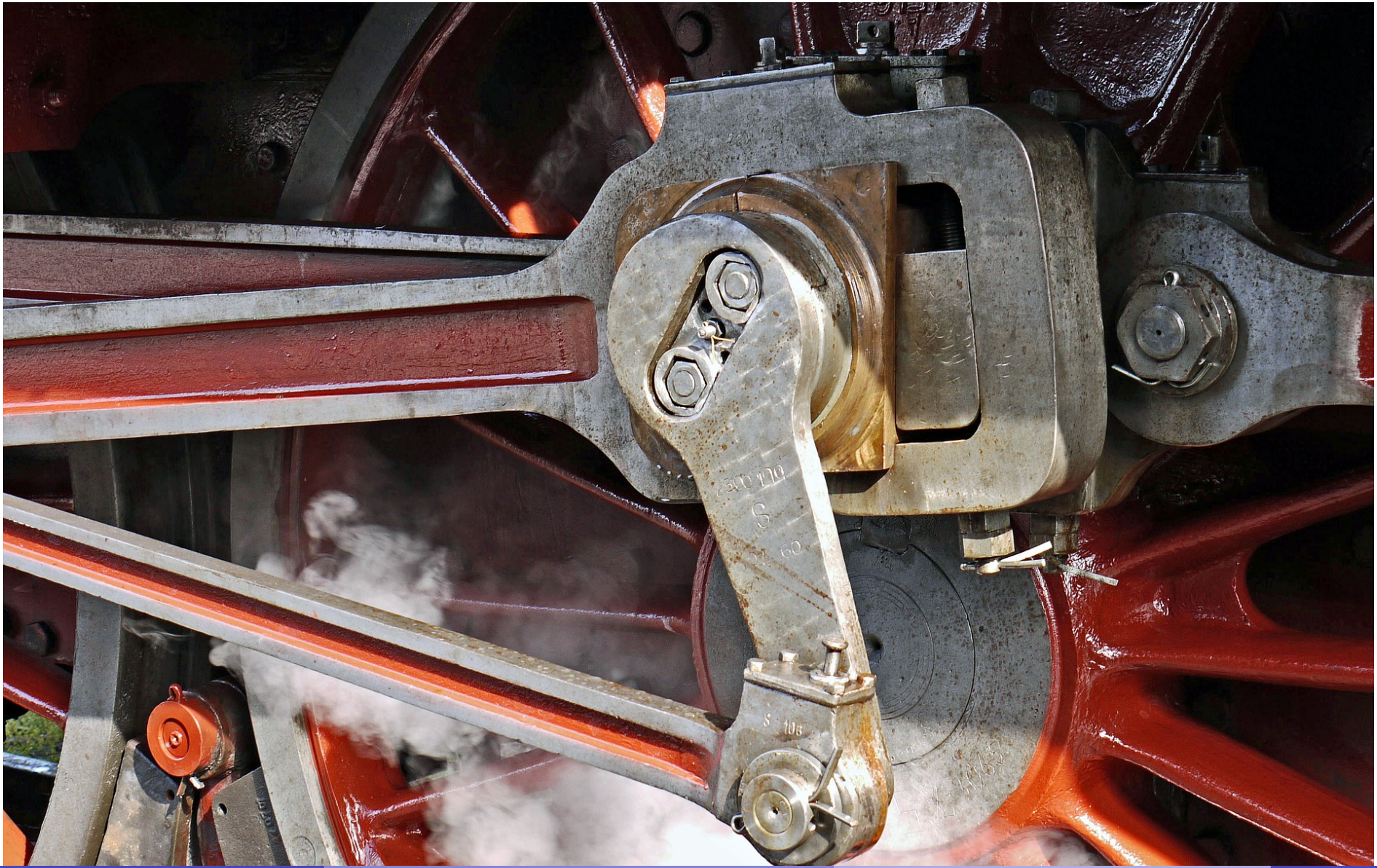
DELETE

```
$country->delete;
```

SELECT

```
$country = $schema->resultset('Country')->search( { ... } );
```

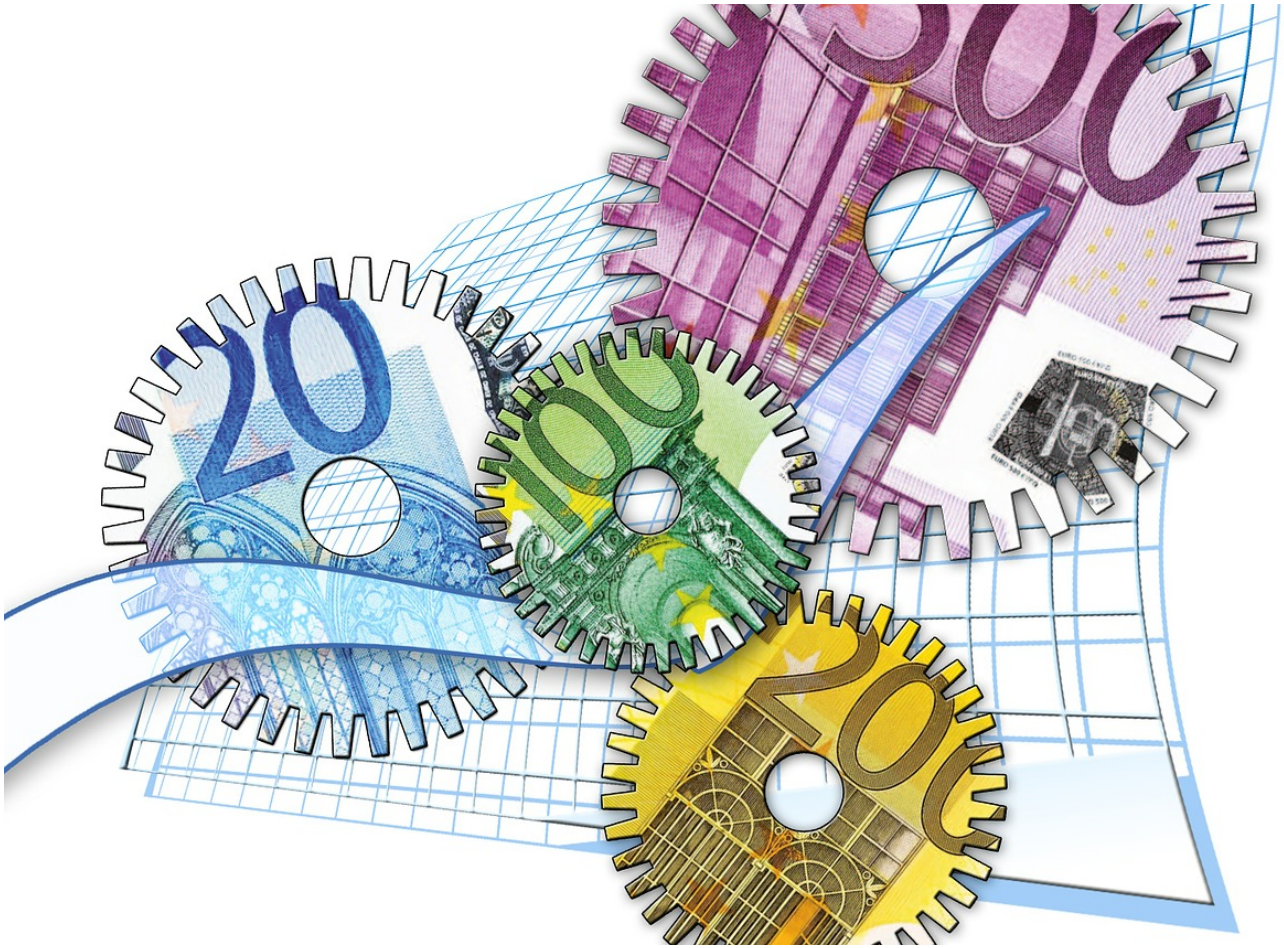
ResultSet-Klassen



DBIx::Class Vorteile

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

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

Interchange

Interchange

- CGI
- Embedded SQL
- ~~Modern Perl~~

eCommerce Innovation 2013

Interchange6::Schema

- Candy
- Individuelle ResultSet-Elternklasse
- Komponenten/Helpers
- Tests mit Test::Roo
- ...

Interchange6::Schema

Fazit

- Mehr als ein ORM
- ResultSet
- Ökosystem
- Dokumentation

DBIx::Class Mengenlehre

Arthur Axel "fREW" Schmidt

Set Based DBIx::Class

DBIx::Class Mengenlehre

- Verkettung
- Relationship Traversal
- Subqueries
  - Korrelierte Subqueries

ResultSet anwenden

- Einfache SQL-Abfrage
- Entsprechende DBIx::Class-Suche

Einfache SQL-Abfrage

Liste der Vorträge:

```
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'
```

Einfache SQL-Abfrage

Mit 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
- Korrelierte Subqueries

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
    }
  }
 );
}
```

Vordefinierte Suche: Produktvarianten

```
has_many
  variants => "Interchange6::Schema::Result::Product",
  { "foreign.canonical_sku" => "self.sku" },
  { cascade_copy            => 0, cascade_delete => 0 };
```

Vordefinierte Suche: Produktvarianten

```
SELECT "me"."sku", "me"."manufacturer_sku", "me"."name",
"me"."short_description", "me"."description", "me"."price",
"me"."uri", "me"."weight", "me"."priority", "me"."gtin",
"me"."canonical_sku", "me"."active",
"me"."inventory_exempt", "me"."combine", "me"."created",
"me"."last_modified",
(SELECT COUNT( * ) FROM "products" "variants_alias"
WHERE ( "variants_alias"."canonical_sku" = "me"."sku" ))
FROM "products" "me"
```

Korrelierte Subqueries - Interchange6::Schema

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

  return $self->search(
    undef,
    {
      '+columns' => {
        status => $self->correlate('statuses')->rows(1)
          ->order_by('!created,!order_status_id')
          ->get_column('status')
          ->as_query,
        },
      }
  );
}
```

Korrelierte Subqueries - Perl Dancer Conference

```
get '/admin/tickets' => require_role admin => sub {
  my $tokens = {};

  $tokens->{title} = "Tickets Sold";

  my $orders = rset('Order');

  $tokens->{orders} = $orders->search(
    {
      payment_status => 'paid',
    },
    {
      prefetch => 'user',
      order_by => 'orders_id',
    }
  )->with_status;

  template 'admin/tickets', $tokens;
```

Relationen

- Relationship Traversal
- Spezifische JOIN-Bedingungen

Relationship Traversal

Liste der besuchten Vorträge eines Teilnehmers:

```
$tokens->{attending} =
  $user->related_resultset('attendee_talks')
    ->search_related( 'talk',
      { 'talk.conferences_id' => setting('conferences_id')
    ->order_by('talk.title');
```

Spezifische JOIN-Bedingungen

Relation für **alle** Adressen eines Benutzers:

```
__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 },
);
```

Ausbaufähig

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

Beispiel für Zusatzattribute

- 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' ]
    )
);
```

Class::C3

Zusatzattribut setzen

```
sub BUILD {
  my $plugin = shift;
   weaken ( my $weak_plugin = $plugin );
  $plugin->app->add_hook(
    Dancer2::Core::Hook->new(
      name => 'before',
      code => sub {
        my $user = $weak_plugin->logged_in_user || undef;
        if ( $user ) {
          $user = $weak_plugin->shop_user->find(
            {
              username => $user->{username}
            }
           );
        }
        $weak_plugin->shop_schema->set_current_user($user);
      },
    )
```

Beispiel: Preisberechnung

Beispiel: Preisberechnung

- Preislisten
- Aktionen
- Rabatte

Tabelle price_modifiers

```
primary_column price_modifiers_id => {
    data_type          => "integer",
    is_auto_increment => 1,
};
column sku => { data_type => "varchar", size => 64 };
column quantity =>  { data_type => "integer", default_value
column roles_id =>  { data_type => "integer", is_nullable =
column price => {
    data_type      => "numeric",
    size           => [ 21, 3 ],
};
column start_date => {
    data_type      => "date",
    is_nullable    => 1,
};
column end_date => {
    data_type      => "date",
    is_nullable    => 1,
```

Preisberechnung

Suchbedingung für Einträge in `price_modifiers`:

```
my $search_cond = {
    'start_date' => [ undef, { '<=', $today } ],
    'end_date'   => [ undef, { '>=', $today } ],
    'quantity'   => { '<=' => $args->{quantity} },
    'roles_id'   => undef,
};
```

Preisberechnung

Bedingung für Rollen des Benutzers einfügen:

```
if ( my $user = $schema->current_user ) {
    $search_cond->{roles_id} = [
        undef,
        {
            -in => $schema->resultset('UserRole')
                ->search( { users_id => $user->id } )
                ->get_column('roles_id')->as_query
            }
        ];
    }
```

Schemas vererben: Anwendungsbeispiele

- Generisches Schema, z.B. `Interchange6::Schema`
  - `PerlDance::Schema`
  - `DanceShop::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' ],
);
```

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
};
```

Umstellung auf Candy

- Umstellung des ganzen Schemas
- Neue Resultklassen
- Neue Spalten, Relationen, ...

Umstellung auf Candy

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

- use base 'DBIx::Class::Core';
+ use TravelDance::Schema::Candy;

-__PACKAGE__->table('users');
+table 'users';
```

Komponenttypen

- Komponenten für Schemaklasse
- Komponenten für Resultklassen
- Komponenten für Resultsetklassen

Schema-Komponenten

```
package Interchange6::Schema;

use strict;
use warnings;

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

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

ResultSet-Komponenten

- Elternklasse anlegen
  `Interchange6::Schema::ResultSet`
- Als voreingestellte ResultSet-Klasse eintragen in
  `Interchange6::Schema`
- Elternklasse in vorhandene ResultSet-Klassen eintragen
  `Interchange6::Schema::ResultSet::User`

Elternklasse anlegen

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

use base 'DBIx::Class::ResultSet';

__PACKAGE__->load_components(
    'Helper::ResultSet::CorrelateRelationship',
    'Helper::ResultSet::Me',
    'Helper::ResultSet::Random',
    'Helper::ResultSet::SetOperations',
    'Helper::ResultSet::Shortcut'
);
```

Als Voreinstellung konfigurieren

```
package Interchange6::Schema;

__PACKAGE__->load_namespaces(
    default_resultset_class => 'ResultSet'
);
```

Vererbung von der Elternklasse

Bereits vorhandene ResultSet-Klassen müssen die eigene ResultSet-Elternklasse übernehmen:

```
package Interchange6::Schema::ResultSet::User;

use base 'Interchange6::Schema::ResultSet';

__PACKAGE__->load_components(
    qw(Helper::ResultSet::CorrelateRelationship)
);
```

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

Nach der Passwortänderung:

- 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 Dokumentation
  - `DBIx::Class::Manual::*`
  - `DBIx::Class::Manual::ResultClass`
  - `DBIx::Class::ResultSet`
  - `DBIx::Class::Relationship::*`
- Helpers von fREW
  - https://metacpan.org/pod/DBIx::Class::Helpers
- 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/hhmongers2017/dbic-pr-de-beamer.pdf`

Fragen

Fragen ?