A train ride through DBIx::Class and its ecosystem

Stefan Hornburg (Racke)

Contents

Searching

The search parameters are in the first parameter of the *search* method.

NOT NULL

```
$rs->search({
    vendor => { '!=', undef }
});
```

Or with helpers:

```
$rs->not_null('vendor');
```

Distinct

If you want to know whether the value for a column is *distinct* and retrieve the value at the *same time*, look at the following example:

```
my $distinct_status = $rs->search({}, {
        columns => [ qw/status/ ],
        group_by => [ qw/status/ ],
});

if ( $distinct_status->count == 1 ) {
    my $ol_status = $distinct_status->first->status;
}
```

Comparing columns

We want to find all records where the columns `username` and `replaced_by` contain the same value. Here is the corresponding SQL query:

```
select * from userdb where username = replaced_by;
```

We can't use a bareword on the right hand side in our search parameters, so we have to resort to use the `-ident` operator:

```
my $user_rs = $schema->resultset('User');

my $circular = $user_rs->search({
    username => { -ident => 'replaced_by' },
});
```

Sorting

```
my $sorted_rs = $schema->resultset('Company')->search(
   {},
   { order_by => { -asc => 'name' } },
);
```

Creating schema from existing database

You can use `dbicdump` to create a "boilerplate" schema from an existing database.
Install first DBIx::Class::Schema::Loader, e.g. with `cpanm`:

```
cpanm DBIx::Class::Schema::Loader
```

Now you run `dbicdump`:

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

Add database credentials (username and password):

```
dbicdump -o dump_directory=/home/dance/TravelDance/lib \
         TravelDance::Schema \
         dbi:Pg:dbname=sympa sympa "nevairbe"
```

You can also include useful components:

```
dbicdump -o dump_directory=/home/dance/TravelDance/lib
         -o components='["InflateColumn::DateTime"]'
         TravelDance::Schema
         dbi:Pg:dbname=perldance
```

Remove comments generated by DBIx::Class::Schema::Loader.

```
perl -i -ne 'print if $_ !~ /(Created by DBIx::)|(DO NOT MODIFY)|(You can replace)/;'  `find lib -na
```

Debugging

```
export DBIC_TRACE=1
```

Result and ResultSet classes

Set-based DBIx::Class

Excellent article from Arthur Axel "fREW" Schmidt explaining the features of the ResultSet class: `http://www.perladvent.org/2012/2012-12-21.html`

Reaching out to another ResultSet class

If you want to reach out from a method in a result or resultset class to another resultset class, you can do the following:

```
my $rset_message = $self->result_source->schema->resultset("Message");
```

Result classes

Updating result objects

Sometimes you want to update your result object from the database record, e.g. after creating a database record to get the values of columns with are automatically updated such as timestamps:

```
$user->create({username => 'nevairbe@linuxia.de'});
$user->discard_changes;
print $user->last_modified, "\n";
```

Flatten a object

This turns the object into a hash:

```
my %user_hash = $user->get_inflated_columns;
```

Predefined searches

Predefined searches encapsulate business logic inside the schema. If you use them in the consumers of the schema, you can change the business logic without changing the code of the consumes.

Example:

```
sub not_amazon {
    my ($self) = @_;

    my $user_rs = $self->search({
        username => { -not_like => '%@marketplace.amazon%' },
    });
```

```
    return $user_rs;
}
```

Relationships

CASCADE actions

Indices

Add indices to your result classes like that:

```
sub sqlt_deploy_hook {
    my ($self, $sqlt_table) = @_;
    $sqlt_table->add_index(name => 'idx_product_canonical_sku',
                           fields => ['canonical_sku']);
}

sub sqlt_deploy_hook {
    my ($self, $sqlt_table) = @_;
    $sqlt_table->add_index(name => 'idx_product_canonical_sku',
                           fields => ['canonical_sku']
                           options => { where => q{canonical_sku <> ''},
                          );
}
```

Components

DBIx::Class uses Class::C3::Componentised to allow mix-ins (components) to be added to the various classes that together comprise the schema definition.

There are three different types of components:

- *Schema* class components
- *Result* class components
- *ResultSet* class components

Many components exist for all of these three kinds.

Schema Components

You can add components like helpers into your main schema class with the method *load_ components*:

```
package Interchange6::Schema;

use strict;
use warnings;

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

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

Using Result Class Components

In a single result class:

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

use TravelDance::Schema::Candy -components => [qw(
    InflateColumn::DateTime
    PassphraseColumn
    TimeStamp
)];
```

In order to add a result class component to *all* result classes you can add these to the list in `TravelDance::Schema::Candy`'s `parse_arguments`.

ResultSet Components

In order to be able to add a component to all ResultSet classes in the schema *Interchange6::Schema* we need to do:

1. create parent class *Interchange6::Schema::ResultSet*
2. set as default resultset class in *Interchange6::Schema*
3. use for custom resultset classes *Interchange6::Schema::ResultSet::User*

Create parent class

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

use strict;
use warnings;

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

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

Set as default ResultSet class

```
package Interchange6::Schema;

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

Exceptions

Throw an exception in your result class:

```
unless ( $self->valid_to > $self->valid_from ) {
    $schema->throw_exception("valid_to is not later than valid_from");
}
```

Helpers

Arthur Axel ”fREW” Schmidt

Adding accessors to the schema

Vital information utilized in the schema and its methods isn't always available in the database, especially if you use it within web applications.

For example, we need to know which user is currently logged in to determine

- product prices
- whether to restrict searches

So we add accessors to the `Schema.pm` module:

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

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

In our web application, we are using the accessor in the following cases:

- start of processing a HTTP request
- user logs in
- user logs out

Deployment Handler

In order to start with deployment handler, add a version number to your main schema module:

```
package TravelDance::Schema;

=head1 NAME

TravelDance::Schema - Database schema class loader for TravelDance

=cut

use warnings;
use strict;

use base 'DBIx::Class::Schema';
use TravelDance::Schema::Populate::CountryLocale;

our $VERSION = 1;
```

To avoid unnecessary headaches, please use consecutive natural numbers.

Scripts

A number of useful scripts:

dh-prepare-version-storage prepares table for DH

dh-install-version-storage install and populates table for DH

dh-prepare-upgrade prepares upgrade

dh-upgrade performs upgrade

Adding relationships

Please make sure that the definition of the foreign key matches the primary key in the other table in terms of data type, length and default value.

```
DBI Exception: DBD::mysql::db do failed: Can't create table 'linuxia.#sql-6f14_e'
(errno: 150) [for Statement "ALTER TABLE orderline CHANGE COLUMN username username
varchar(255) NOT NULL, ADD INDEX orderline_idx_username (username), ADD CONSTRAINT
orderline_fk_username FOREIGN KEY (username) REFERENCES userdb (username) ON DELETE SET NULL"]
```

An other reason for such an error are rows that violate the foreign key, e.g. a value for *username* in *orderline*, which doesn't appear in *orderline*.

Linuxia Wiki

Stefan Hornburg (Racke)
A train ride through DBIx::Class and its ecosystem

wiki.linuxia.de