A DBIC::Debacle

Where Relationships sub-tly Cascade Out of Control

Julien Fiegehenn (simbabque)
Deutscher Perl/Raku-Workshop 2023
27. Februar - 01. März 2023, Frankfurt

Recap: relationship types

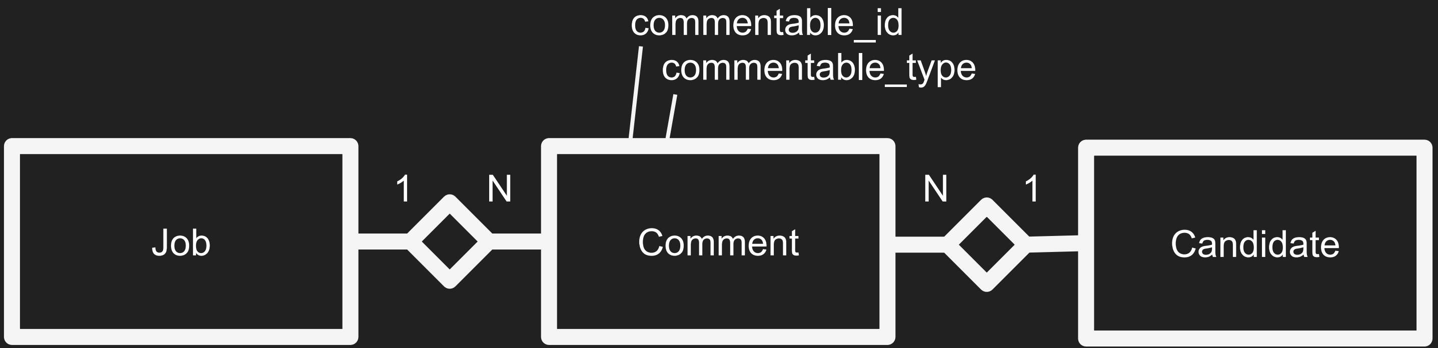
`belongs_to`	“many to one”, a book was written by an author
`has_many`	“one to many”, an author has written many books
`might_have`	“optional one to one”, an author might have a pseudonym
`has_one`	“one to one”, a book has exactly one ISBN
`many_to_many`	“many to many”, many actors play many roles

Recap: how to use

Arguments: `$accessor_name`, `$related_class`, `$their_fk_column|\%cond|\@cond|\&cond?`, `\%attrs?`

```
My::DBIC::Schema::Author->has_many(
  books =>
  'My::DBIC::Schema::Book',
  { 'foreign.author_id' => 'self.id' },
);
```

What are we trying to do?



How are we doing it?

```
__PACKAGE__->has_many(
    'comments' => 'My::DBIC::Comment',         # accessor => related class
    {
        'foreign.commentable_id' => 'self.id'  # \%condition
    },
    {                                          # \%attributes
        'where' => {
            'commentable_type' => 'job'        # a where condition
        }
    }
);
```

This now works great

This Perl code…

```
my $job = $schema->resultset('Jobs')->first;
$job->comments->first;
```

Produces this SQL…

```
SELECT *
  FROM comments
 WHERE commentable_id = '1' AND commentable_type = 'job'
```

But this does not!

This Perl code…

```
$schema->resultset('Jobs')
   ->search( {}, { prefetch => 'comments' } )->first
   ->comments->first;
```

Produces this SQL…

```
SELECT *
  FROM jobs me
  LEFT JOIN comments comments
    ON comments.commentable_id = me.id
```

Is it a bug if it’s been known since 2011?

In December 2010: https://rt.cpan.org/Public/Bug/Display.html?id=63709

“Bug #63709 for DBIx-Class: relationship where attrs are ignored when prefetching a rel”

Is it a bug if it’s been known since 2011?

In December 2010: https://rt.cpan.org/Public/Bug/Display.html?id=63709

“Bug #63709 for DBIx-Class: relationship where attrs are ignored when prefetching a rel”

Ribasushi, August 2011: “[...] it is an (obviously) known issue, with an annoyingly involved fix. Due to the limited utility of the where attr, this bug has not been as of yet"

DBIx::Class::Relationship::Base - custom join cond

To specify joins which describe more than a simple equality of column values, the custom join condition coderef syntax can be used.

```
My::DBIC::Job->has_many(
 'comments' => 'My::DBIC::Comments',
 sub {
   my $args = shift;
    return {
     "$args->{foreign_alias}.commentable_id"    => { -ident => "$args->{self_alias}.id" },
     "$args->{foreign_alias}.commentable_class" => 'job',
   };
 }
);
```

DBIx::Class::Relationship::Base - custom join cond

To specify joins which describe more than a simple equality of column values, the custom join condition coderef syntax can be used.

```
sub {
  my $args = shift;
   return {
    "$args->{foreign_alias}.commentable_id"    => { -ident => "$args->{self_alias}.id" },
    "$args->{foreign_alias}.commentable_class" => 'job',
  };
}
```

```
SELECT *
 FROM jobs me
 LEFT JOIN comments comments
   ON comments.commentable_id = me.id AND comments.commentable_type = 'job'
```

The full sub with all the trimmings

```perl
sub {
 my $args = shift;
 return (
   {
     "$args->{foreign_alias}.commentable_id"   => { -ident => "$args->{self_alias}.id" },
     "$args->{foreign_alias}.commentable_type" => 'job'
   },
   ! $args->{self_result_object} ? () : {
     "$args->{foreign_alias}.commentable_id"   => $args->{self_result_object}->id(),
     "$args->{foreign_alias}.commentable_type" => 'job'
   },
   ! $args->{foreign_values} ? () : {
     "$args->{self_alias}.id" => $args->{foreign_values}{commentable_id},
   }
 );
}
```

So now we’re done, right?

So now we’re done, right?

No.

Recap: how to use

Arguments: `$accessor_name, $related_class, $their_fk_column|\%cond|\@cond|\&cond?, \%attrs?`

```
My::DBIC::Schema::Author->has_many(
  books =>
  'My::DBIC::Schema::Book',
  { 'foreign.author_id' => 'self.id' },
  { cascade_delete => 1,
    cascade_copy   => 1 },
);
```

Default values for cascading

	has_many	has_one	might_have
cascade_copy	on		
cascade_update		on	on
cascade_delete	on	on	on

(https://metacpan.org/pod/DBIx::Class::Relationship::Base#attributes)

Cascade Failure

```
package # hide from PAUSE
   DBIx::Class::Relationship::HasMany;
sub has_many {
 my ($class, $rel, $f_class, $cond, $attrs) = @_;
 # ...
 my $default_cascade = ref $cond eq 'CODE' ? 0 : 1;
 $class->add_relationship($rel, $f_class, $cond, {
   accessor => 'multi',
   join_type => 'LEFT',
   cascade_delete => $default_cascade,
   cascade_copy => $default_cascade,
   is_depends_on => 0,
   %{$attrs||{}}
 });
}
```

The full relationship

```perl
My::DBIC::Job->has_many(
 'comments' => 'My::DBIC::Comments',
 sub {
   my $args = shift;
   return (
     {
       # ...
     }
   );
 },
 {
   cascade_delete => 1,
   # ...
 }
);
```

Conclusion

- If you have complex join conditions, **do not use WHERE**
- Always be explicit with cascade settings
- If you can’t read the docs, read the code!

Thank you!

Julien Fiegehenn

simbabque@cpan.org

@simbabque

simbabque

Questions?