

Datenbanken und Perl

Von DBI bis **DBIx::Class**

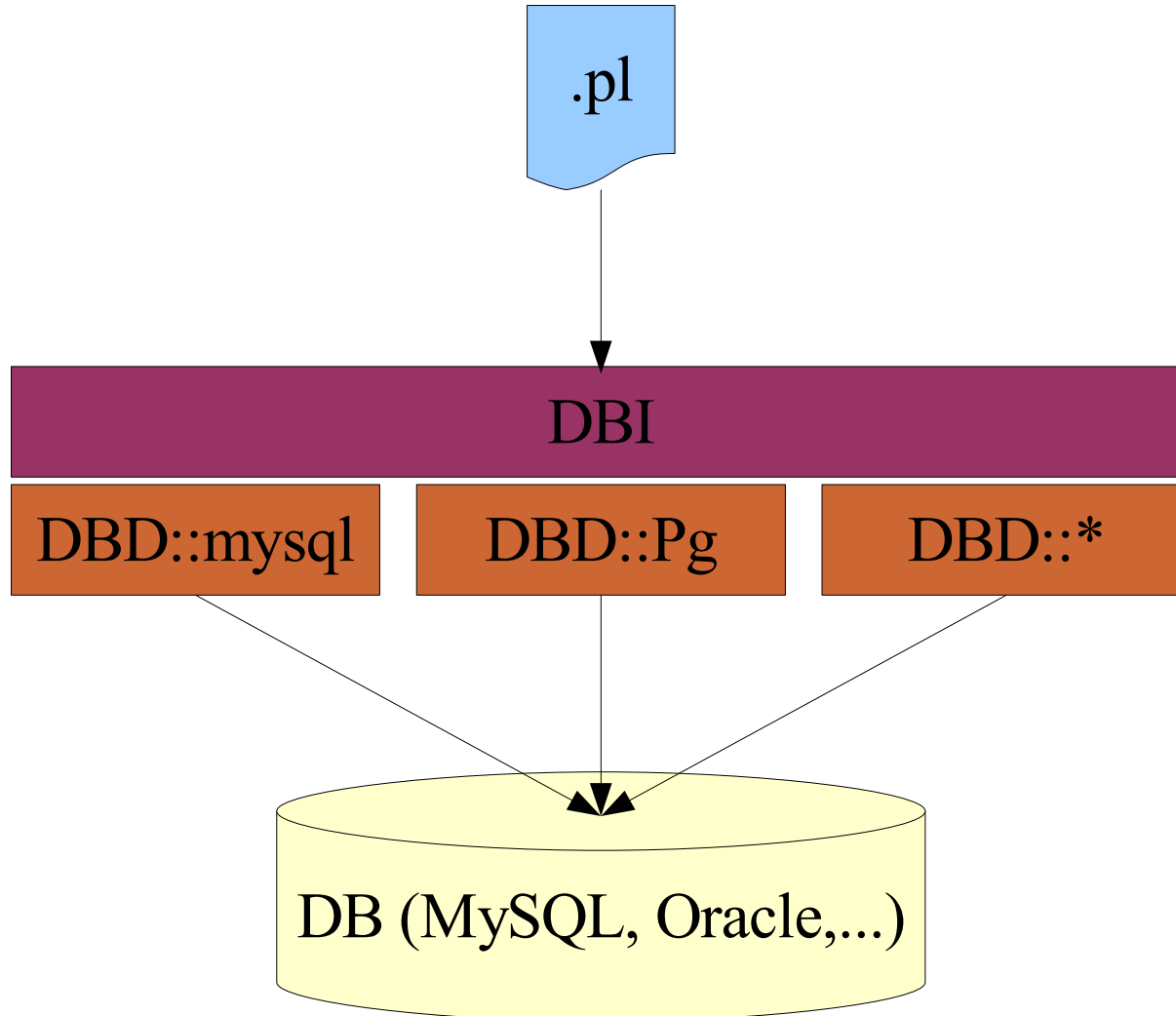
- Perl mit Datenbanken verbinden ist wirklich einfach.
- Dank DBI (DataBase Interface) einheitliches Gesicht für alle Datenbanken

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

```
use strict;  
use warnings;
```

```
use DBI;
```

Datenbanken und Perl



- Beim Umstieg im Idealfall nur 1 Zeile austauschen:

```
my $dbh = DBI->connect ("DBI:mysql:dbname:host") ;
```

vs

```
my $dbh = DBI->connect ("DBI:Pg:dbname:host") ;
```

- Interaktion mit SQL...

```
my $stmt = qq~SELECT * FROM tabelle~;  
my $sth  = $dbh->prepare( $stmt ) or  
          die $dbh->errstr;  
  
$sth->execute or die $dbh->errstr;  
  
while( my @row = $sth->fetchrow_array ) {  
    print "@row\n";  
}
```

- Interaktion mit SQL...

```
my $stmt = qq~SELECT * FROM tabelle~;
my $sth  = $dbh->prepare( $stmt ) or
           die $dbh->errstr;

$sth->execute or die $dbh->errstr;

while( my @row = $sth->fetchrow_array ) {
    print "@row\n";
}
```

- Interaktion mit SQL...

```
my $stmt = qq~SELECT * FROM tabelle~;  
my $sth  = $dbh->prepare( $stmt ) or  
          die $dbh->errstr;
```

```
$sth->execute or die $dbh->errstr;
```

```
while( my @row = $sth->fetchrow_array ) {  
    print "@row\n";  
}
```

- Interaktion mit SQL...

```
my $stmt = qq~SELECT * FROM tabelle~;
my $sth  = $dbh->prepare( $stmt ) or
           die $dbh->errstr;

$sth->execute or die $dbh->errstr;

while( my @row = $sth->fetchrow_array ) {
    print "@row\n";
}
```

- Zur Sicherheit die ?-Notation verwenden...

```
my $stmt = qq~SELECT * FROM tabelle WHERE  
                username = ?~;
```

```
my $sth = $dbh->prepare( $stmt ) or  
                die $dbh->errstr;
```

```
$sth->execute( "Tim O'Reilly" ) or  
                die $dbh->errstr;
```

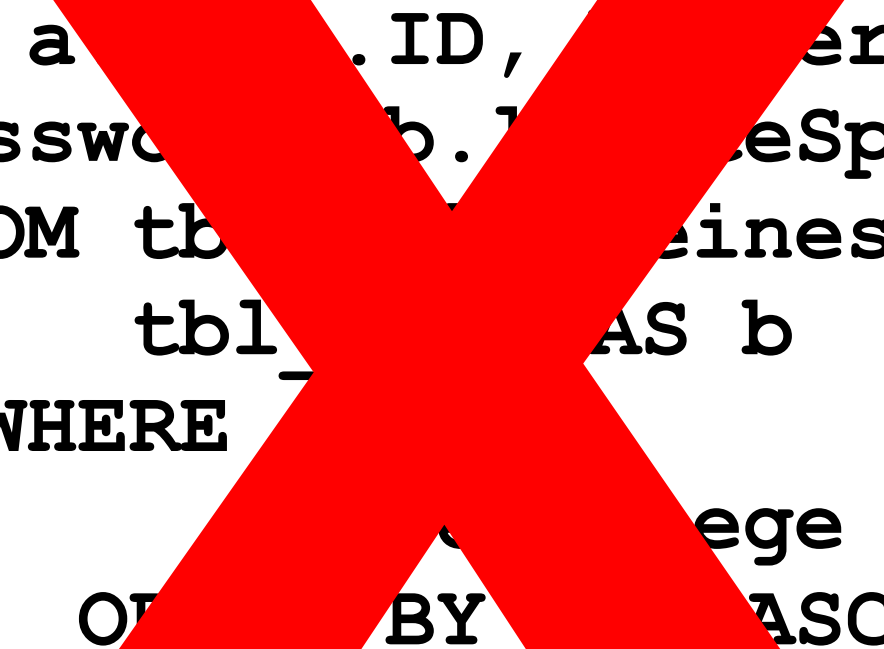
Was ist DBIx::Class?

```
SELECT a.*, b.ID, b.Username,  
       b.Password, b.letzteSpalte  
FROM tbl_allgemeines AS a,  
     tbl_User AS b  
WHERE a.ID = b.ID AND  
       a.beitraege = 199  
ORDER BY a.IDASC
```



Was ist DBIx::Class?

```
SELECT a.ID, a.Username,  
       b.Password, b.ID AS bID, b.Spalte  
FROM tbl1 AS a, tbl2 AS b  
WHERE a.ID = b.ID AND  
       a.Age = 199  
ORDER BY a.ID ASC
```



Was ist DBIx::Class?


```
my $results = $obj->resultset( 'Allgemeines' )  
    ->search(  
        #...  
    );
```

Doch von vorne...

Was ist DBIx::Class

<http://de.wikipedia.com/wiki/ORM>:

„**Objektrelationale Abbildung** (englisch *object-relational mapping*, ORM) ist eine Technik der Softwareentwicklung, mit der ein in einer objektorientierten Programmiersprache geschriebenes Anwendungsprogramm seine Objekte in einer relationalen Datenbank ablegen kann. Dem Programm erscheint die Datenbank dann als objektorientierte Datenbank, was die Programmierung erleichtert. Implementiert wird diese Technik normalerweise über Klassenbibliotheken, wie beispielsweise [...]“
DBIx::Class für die Programmiersprache Perl.



DBIx::Class ist das
„Standard“-ORM
für Perl

Einträge holen

```
my ($spieler) = $schema->resultset( 'Spieler' )
    ->search({
    SpielerID => 3,
});

print $spieler->SpielerName;
```

Einträge aktualisieren

```
my ($spieler) = $schema->resultset( 'Spieler' )
                                ->search({
    SpielerID => 3,
});

print $spieler->SpielerName;

$spieler->update({
    SpielerName => 'Ronaldo',
});
```





Neue Einträge in der Datenbank

```
my $neuer_spieler =  
  $schema->resultset( 'Spieler' )  
    ->create({  
      SpielerName => 'Beckham',  
    });
```

Einträge löschen

```
$schema->resultset( 'Spieler' )  
    ->search({  
        SpielerID => 3,  
    }) ->delete;
```

Eine Tabelle

BundesligaTabelle ▼	
	Platz: INTEGER
	Verein: VARCHAR
	Punkte: INTEGER
	Tore: VARCHAR

Eine Tabelle

```
package My::DB::Result::Tabelle;
```

```
use base qw(DBIx::Class);
```

```
__PACKAGE__->load_components(  
    qw/PK::Auto Core/  
);
```

```
__PACKAGE__->table(  
    'BundesligaTabelle',  
);
```

```
__PACKAGE__->add_columns(qw/  
    Platz  
    Punkte  
    Tore  
    VereinID  
/);
```

```
__PACKAGE__->set_primary_key('Platz');
```

```
1;
```

BundesligaTabelle	
Platz: INTEGER	
Verein: VARCHAR	
Punkte: INTEGER	
Tore: VARCHAR	

Eine Tabelle

```
package My::DB::Result::Tabelle;
```

```
use base qw(DBIx::Class);
```

```
__PACKAGE__->load_components(  
    qw/PK::Auto Core/  
);
```

```
__PACKAGE__->table(  
    'BundesligaTabelle',  
);
```

```
__PACKAGE__->add_columns(qw/  
    Platz  
    Punkte  
    Tore  
    VereinID  
/);
```

```
__PACKAGE__->set_primary_key('Platz');
```

```
1;
```

BundesligaTabelle	
Platz: INTEGER	
Verein: VARCHAR	
Punkte: INTEGER	
Tore: VARCHAR	

Eine Tabelle

```
package My::DB::Result::Tabelle;
```

```
use base qw(DBIx::Class);
```

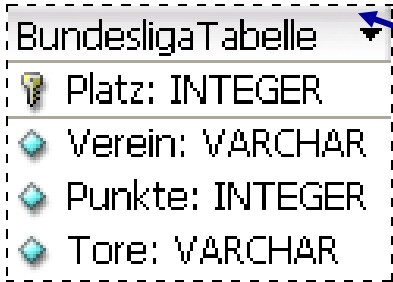
```
__PACKAGE__->load_components(  
    qw/PK::Auto Core/  
);
```

```
__PACKAGE__->table(  
    'BundesligaTabelle',  
);
```

```
__PACKAGE__->add_columns(qw/  
    Platz  
    Punkte  
    Tore  
    VereinID  
/);
```

```
__PACKAGE__->set_primary_key('Platz');
```

```
1;
```



BundesligaTabelle	
🔑 Platz: INTEGER	
🔹 Verein: VARCHAR	
🔹 Punkte: INTEGER	
🔹 Tore: VARCHAR	

Eine Tabelle

```
package My::DB::Result::Tabelle;
```

```
use base qw(DBIx::Class);
```

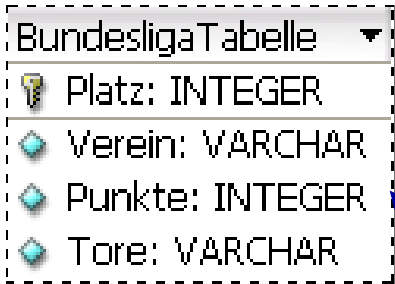
```
__PACKAGE__->load_components(  
    qw/PK::Auto Core/  
);
```

```
__PACKAGE__->table(  
    'BundesligaTabelle',  
);
```

```
__PACKAGE__->add_columns(qw/  
    Platz  
    Punkte  
    Tore  
    VereinID  
/);
```

```
__PACKAGE__->set_primary_key('Platz');
```

```
1;
```



Column Name	Column Type
Platz	INTEGER
Verein	VARCHAR
Punkte	INTEGER
Tore	VARCHAR

Eine Tabelle

```
package My::DB::Result::Tabelle;
```

```
use base qw(DBIx::Class);
```

```
__PACKAGE__->load_components(  
    qw/PK::Auto Core/  
);
```

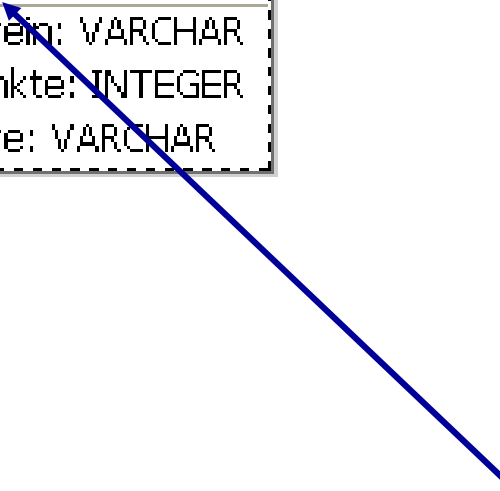
```
__PACKAGE__->table(  
    'BundesligaTabelle',  
);
```

```
__PACKAGE__->add_columns(qw/  
    Platz  
    Punkte  
    Tore  
    VereinID  
/);
```

```
__PACKAGE__->set_primary_key('Platz');
```

```
1;
```

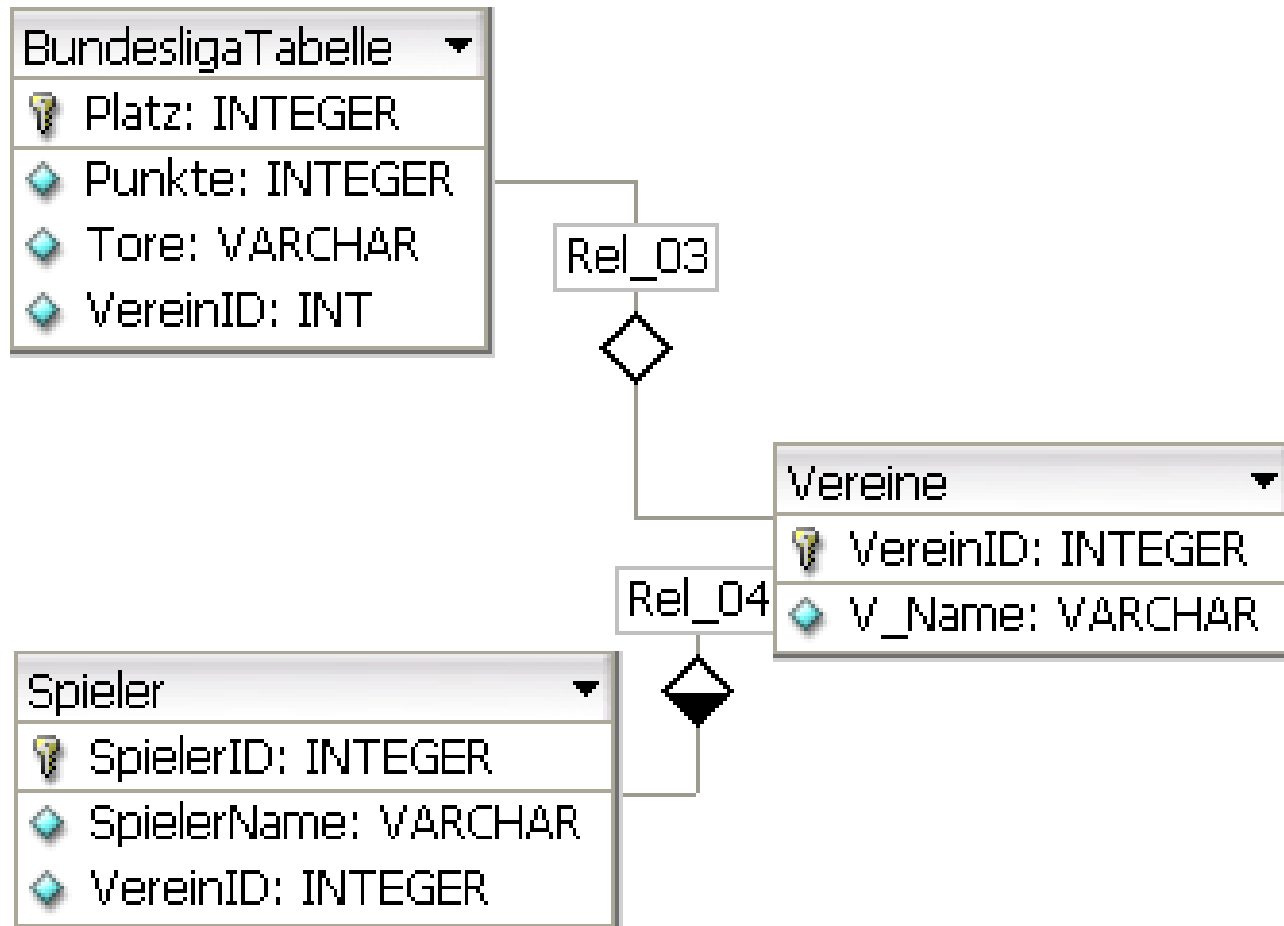
BundesligaTabelle	
Platz: INTEGER	
Verein: VARCHAR	
Punkte: INTEGER	
Tore: VARCHAR	



Beziehungen

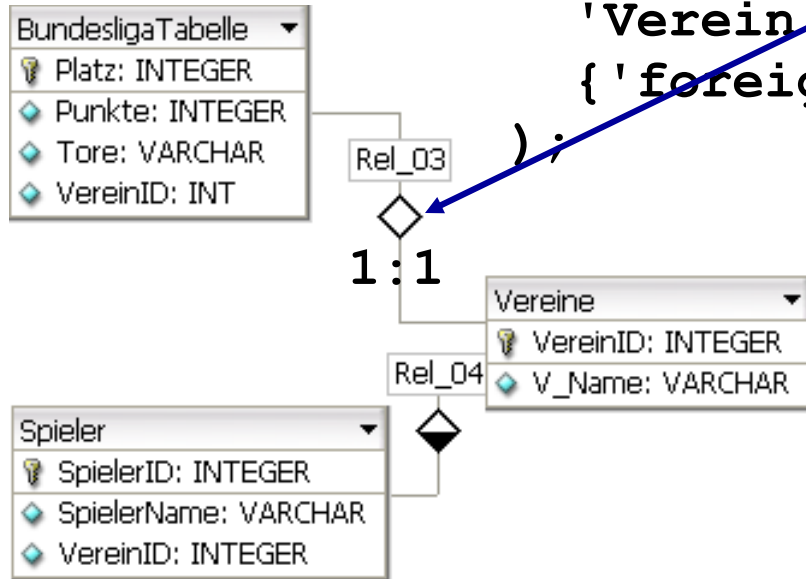
- DBIx::Class unterstützt auch die verschiedenen Beziehungen, die Tabellen haben können...

Beziehungen



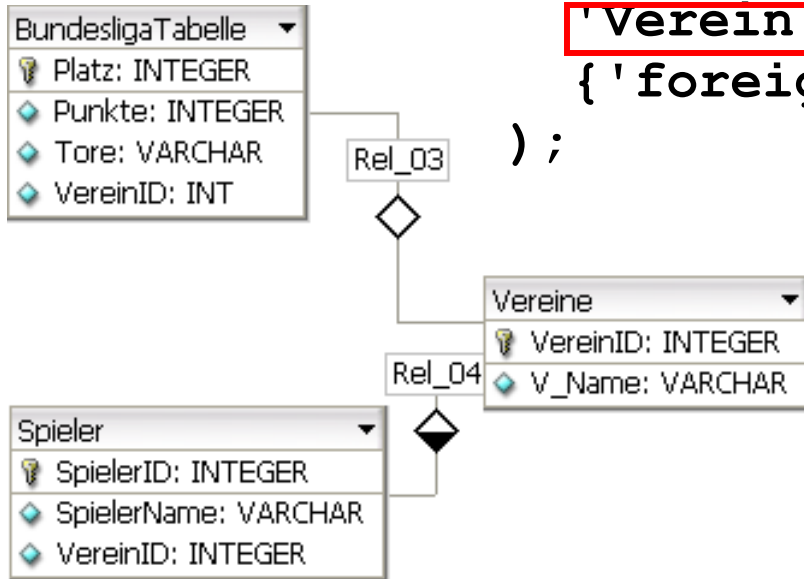
Beziehungen

```
package My::DB::Result::Tabelle;  
# ...  
__PACKAGE__->has_one(  
    'Verein' => 'My::DB::Verein',  
    {'foreign.VereinID' => 'self.VereinID'}  
);
```



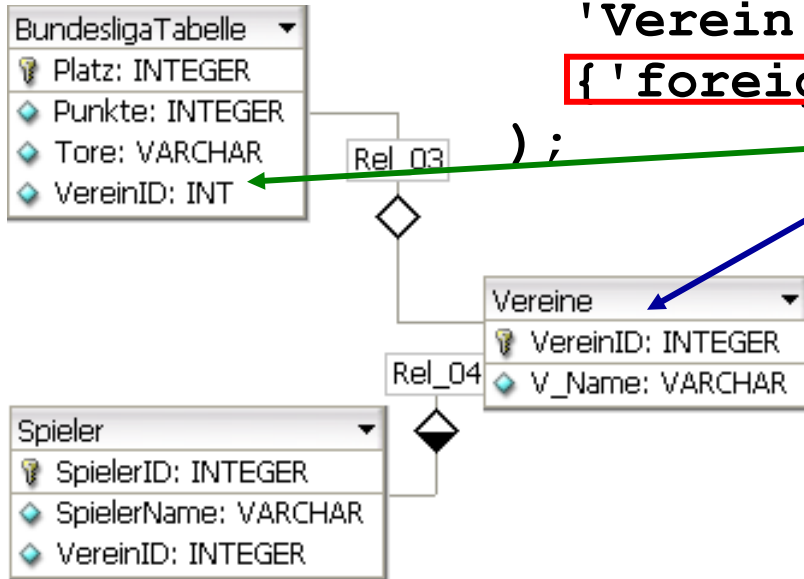
Beziehungen

```
package My::DB::Result::Tabelle;  
# ...  
__PACKAGE__->has_one(  
    'Verein' => 'My::DB::Verein',  
    {'foreign.VereinID' => 'self.VereinID'}  
);
```

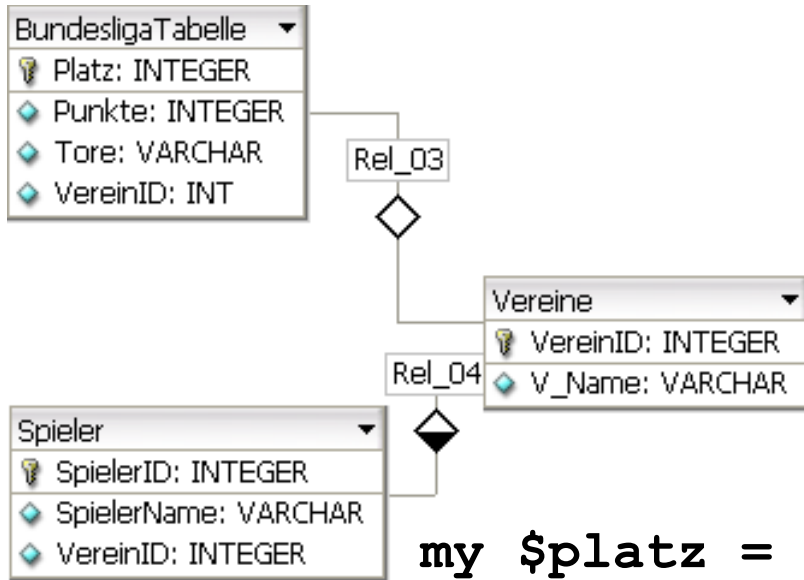


Beziehungen

```
package My::DB::Result::Tabelle;  
# ...  
__PACKAGE__->has_one(  
  'Verein' => 'My::DB::Verein',  
  { 'foreign.VereinID' => 'self.VereinID' }  
);
```



Beziehungen

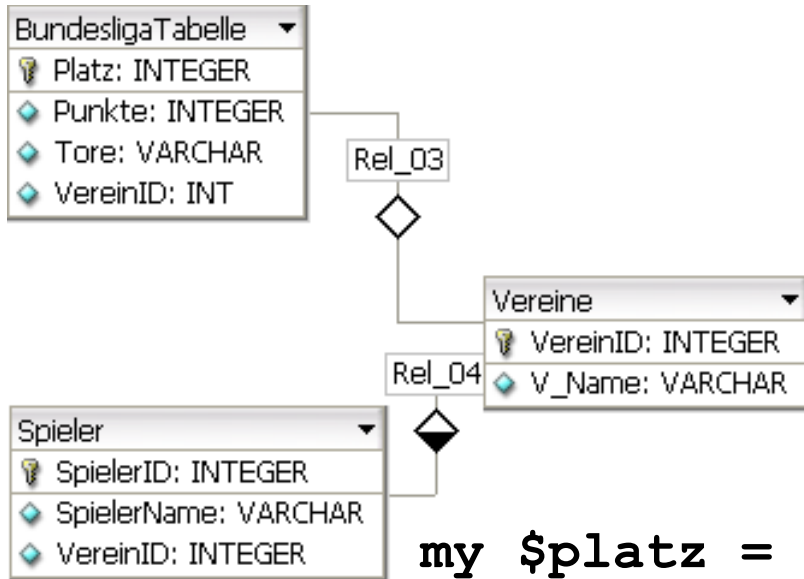


```
my $platz = $schema->resultset( 'Tabelle')
                    ->search({
                        Platz => 1,
                    });

print $platz->Verein->V_Name;
```

Beziehungen

```
__PACKAGE__->has_one(  
  'Verein' => 'My::DB::Verein',  
);
```



```
my $platz = $schema->resultset( 'Tabelle')  
    ->search({  
      Platz => 1,  
    });  
  
print $platz->Verein->V_Name;
```

Beziehungen

- Immer gleicher Aufbau
- Methodename, Klasse, Referenz
- 1:1 \Rightarrow has_one
- N:1 \Rightarrow belongs_to
- 1:N \Rightarrow has_many
- M:N \Rightarrow many_to_many

Und wie benutze ich das?

```
package My::DB;
```

```
use strict;
```

```
use warnings;
```

```
→ use base qw(DBIx::Class::Schema);
```

```
__PACKAGE__->load_namespaces;
```

```
1;
```

Und wie benutze ich das?

```
package My::DB;
```

```
use strict;
```

```
use warnings;
```

```
use base qw(DBIx::Class::Schema);
```

```
→ __PACKAGE__->load_namespaces;
```

```
1;
```

Das Skript...

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

```
use strict;  
use warnings;
```

```
→ use My::DB;
```

```
my $schema = My::DB->connect(...);
```

```
my @entries = $schema->resultset('Bundesliga')->all;
```

```
for my $entry(@entries) {
```

```
    print sprintf "%2d %-12s (ID: %2d) %2d %6s\n",
```

```
        $entry->Platz,
```

```
        $entry->Verein->V_Name,
```

```
        $entry->VereinID,
```

```
        $entry->Punkte,
```

```
        $entry->Tore);
```

```
}
```

Das Skript...

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

```
use strict;  
use warnings;  
use My::DB;
```

```
→ my $schema = My::DB->connect(...);
```

```
my @entries = $schema->resultset('Bundesliga')->all;  
for my $entry(@entries) {  
    print sprintf "%2d %-12s (ID: %2d) %2d %6s\n",  
        $entry->Platz,  
        $entry->Verein->V_Name,  
        $entry->VereinID,  
        $entry->Punkte,  
        $entry->Tore);  
}
```

Das Skript...

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

```
use strict;  
use warnings;  
use My::DB;
```

```
my $schema = My::DB->connect(...);
```

```
→ my @entries = $schema->resultset('Bundesliga')->all;  
for my $entry(@entries) {  
    print sprintf "%2d %-12s (ID: %2d) %2d %6s\n",  
        $entry->Platz,  
        $entry->Verein->V_Name,  
        $entry->VereinID,  
        $entry->Punkte,  
        $entry->Tore);  
}
```

Das Skript...

```
#!/usr/bin/perl

use strict;
use warnings;
use My::DB;

my $schema = My::DB->connect(...);

my @entries = $schema->resultset('Bundesliga')->all;
for my $entry(@entries) {
    print sprintf "%2d %-12s (ID: %2d) %2d %6s\n",
        $entry->Platz,
        $entry->Verein->V_Name,
        $entry->VereinID,
        $entry->Punkte,
        $entry->Tore);
}
```



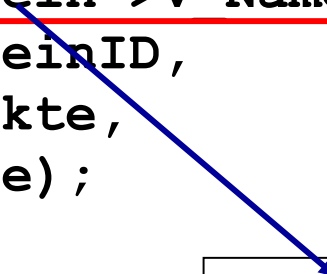

Das Skript...

```
#!/usr/bin/perl

use strict;
use warnings;
use My::DB;

my $schema = My::DB->connect(...);

my @entries = $schema->resultset('Bundesliga')->all;
for my $entry(@entries){
    print sprintf "%2d %-12s (ID: %2d) %2d %6s\n",
        $entry->Platz,
        $entry->Verein->V Name,
        $entry->VereinID,
        $entry->Punkte,
        $entry->Tore);
}
```

'Verein' => 'My::DB::Verein',

Noch weniger Aufwand...

- Wenn DB Fremdschlüssel unterstützt, kann das Schema auch automatisch geladen werden:

Noch weniger Aufwand...

```
package My::DB;
use base qw(DBIx::Class::Schema::Loader);

use strict;
use warnings;

__PACKAGE__->connection(
    'DBI:Pg:Bundesliga:localhost',
    'user',
    'password',
    {
        debug => $ENV{DBIC_TRACE} || 0,
        use_namespaces => 1,
    }
);

__PACKAGE__->load_namespaces;
```

Struktur...

Schema

Result::*

Einzelne Zeile

ResultSet::*

Satz von Zeilen

Schema

- Datenbank-Verbindung
- Evtl. laden des Schemas

Result::*

- Neben der Tabellendefinition kann man hier noch das Verhalten einzelner Spalten beeinflussen:

```
__PACKAGE__->inflate_column(  
    'datumsspalte',  
    {  
        inflate => sub{ scalar localtime( $_[0] ) },  
        deflate => sub{  
            # parsen des datumsstrings und rueckgabe  
            # des timestamps  
        }  
    }  
);
```

Result::~*

- Oder neue „Spalten/Methoden“ für jedes Ergebnis hinzufügen:

```
sub tordifferenz {  
  my ($self) = @_;  
  
  my $diff = $self->geschossene_tore - $self->kassierte_tore;  
  return $diff;  
}
```

ResultSet:*

- Hier können zusätzliche Suchfunktionen definiert werden:

```
sub meisterkandidat {  
    my ($self) = @_;  
  
    return $self->search({  
        Platz => 1,  
    });  
}
```

ResultSet:*

- Hier können zusätzliche Suchfunktionen definiert werden:

```
my $tabelle = $schema->resultset( 'Tabelle' );  
my $meister = $tabelle->search({  
    Platz => 1,  
});
```

```
print $meister->Name;
```

```
my $tabelle = $schema->resultset( 'Tabelle' );  
my $meister = $tabelle->meisterkandidat;  
print $meister->Name;
```

Komplexere Abfragen

```
#SELECT cd.title FROM cd JOIN map ON map.cd = cd.id
#         JOIN genre ON map.genre = genre.id
#         WHERE cd.year = 1995
#         AND genre.name = 'Techno';
```

```
my @all_titles = $schema->resultset( 'CD' )
    ->search( {
    'genre.name' => 'Techno',
    'me.year' => '1995',
},
{
    join => [qw/map genre/],
} )->all;
```

Komplexere Abfragen

```
my @albums = $schema->resultset('Album')->search({
  -or => [
    -and => [
      artist => { 'like', '%Smashing Pumpkins%' },
      title => 'Siamese Dream',
    ],
    artist => 'Starchildren',
  ],
});
```

```
# WHERE ( artist LIKE '%Smashing Pumpkins%'
#         AND title = 'Siamese Dream' )
#       OR artist = 'Starchildren'
```

Fazit

- DBIx::Class ist ein sehr gutes ORM
- Durch die Relationships ist „Navigation“ durch Tabellen möglich...
- Komplexe SQL-Befehle können auch bei DBIx::Class ziemlich komplex werden

- Unterstützt beim Erstellen von Klassen
- Noch am Anfang der Entwicklung

```
#!/usr/bin/perl

use strict;
use warnings;
use FabForce::DBDesigner4::DBIC;

my $dbic = FabForce::DBDesigner4::DBIC->new(
    inputfile => 'test.xml',
    namespace => 'FrOSCon2009',
);
$dbic->create_scheme;
```

Referenzen

- <http://search.cpan.org/dist/DBIx-Class/>
- <http://faq.perl-community.de/bin/view/Wissensbasis/DBIxClassLiteralSQL>
- <http://perlcast.com/2007/06/12/matt-trout-on-dbixclass/>
- <http://www.slideshare.net/ranguard/dbixclass-beginners-presentation>