SQL-Statements erzeugen

Bei der Entwicklung von Perl-Programmen hört man immer wieder, dass es zwei Möglichkeiten gibt, Datenbankabfragen zu machen: Erstens die SQL-Statements in den Code schreiben und mit DBI direkt zu arbeiten und zweitens Objekt-Relationale-Mapper wie DBIx::Class oder Class::DBI zu verwenden. Beides hat Vor- und Nachteile. Während die Arbeit mit DBI schnell und "schlank" ist, ist DBIx::Class relativ langsam, da sehr viele Module geladen werden müssen und die Objekte groß werden. Dafür kann man mit DBIx::Class sehr gut durch die Tabellen "navigieren". Man muss sich im Perl-Code nicht mehr darum kümmern, wie die einzelnen Tabellen miteinander verknüpft sind. Bei DBI ohne zusätzliche Module muss man SQL-Statements schreiben.

Mit SQL::Abstract steht ein Mittelweg zur Verfügung. Das Modul generiert aus Perl-Datentypen ein SQL-Statement, das man dann an DBI übergeben kann (siehe Listing 1).

LIMIT, ORDER BY, ...

SQL::Abstract kann nur einfache Statements erzeugen und die Möglichkeiten für die Angabe von LIMIT oder ORDER BY fehlen komplett. Dafür muss ein weiteres Modul installiert werden: SQL::Abstract::Limit (siehe Listing2).

Das Modul erzeugt immer ein ORDER BY sobald vier Parameter übergeben werden. Ein LIMIT ohne ORDER BY ist nicht möglich - im Gegensatz zu einer Abfrage, die man selbst schreibt. Weiterhin muss noch ein limit_dialect angegeben werden, der für jedes Datenbanksystem die passende LIMIT-Syntax kennt. Wer mit MySQL arbeitet, muss 'LimitXY' wählen (siehe Listing 3).

```
use SQL::Abstract;

my $sql   = SQL::Abstract->new;
my $table = 'testtabelle';
my $cols  = [qw/col1 col2 col3/];
my $where = {
   ID => 129,
};

my ($stmt,@binds) = $sql->select(
    $table, $cols, $where
);

print $stmt, "\n   ", join(" :: ", @binds),"\n";
__END__
C:\>abstract.pl
SELECT col1, col2, col3 FROM testtabelle WHERE ( ID = ? )
   129
```

Listing 1

```
my $sql   = SQL::Abstract::Limit->new( limit_dialect => 'LimitXY' );
my ($stmt,@binds) = $sql->select(
    $table, $cols, $where, ['col1']
);
__END__
C:\>abstract_2.pl
SELECT col1, col2, col3 FROM testtabelle WHERE ( ID = ? ) ORDER BY col1
129
```

Listing 2

```
my $sql   = SQL::Abstract::Limit->new( limit_dialect => 'LimitXY' );
my ($stmt,@binds) = $sql->select(
    $table, $cols, $where, ['col1'], 10
);
__END__
C:\>abstract_2.pl
SELECT col1, col2, col3 FROM testtabelle WHERE ( ID = ? ) ORDER BY col1 LIMIT 10
129
```

Listing 3

```
my $sql   = SQL::Abstract->new;
my $where = {
   ID => 129,
   col5 => 3,
};

my ($stmt,@binds) = $sql->select(
    'test', 'col2', $where
);

print $stmt,"\n";
__END__
C:\>abstract_2.pl
SELECT col2 FROM test WHERE ( ID = ? AND col5 = ? )
```

Listing 4

```
my $sql   = SQL::Abstract->new;
my $where = [
   ID => 129,
   col5 => 3,
];

my ($stmt,@binds) = $sql->select(
    'test', 'col2', $where
);

print $stmt,"\n";
__END__
C:\>abstract_2.pl
SELECT col2 FROM test WHERE ( ( ID = ? ) OR ( col5 = ? ) )
```

Listing 5

Um mit `LIMIT` arbeiten zu können, muss man neben dem vierten Parameter für das `ORDER BY` noch einen fünften Parameter angeben, der den Offset angibt.

Man darf aber nicht vergessen, dass komplexe Abfragen auch mit `DBIx::Class` und `SQL::Abstract` schnell komplex werden. Vor allem am Anfang treten auch immer wieder Probleme damit auf, was denn jetzt Hashreferenzen für eine Auswirkung haben und was Arrayreferenzen bedeuten. Gewisse Sachen, wie zum Beispiel Aufrufe von SQL-Funktionen, müssen auch mit `SQL::Abstract` direkt geschrieben werden, so dass auch weiterhin Kenntnisse über die Funktionen und Möglichkeiten des Datenbanksystems nötig sind.

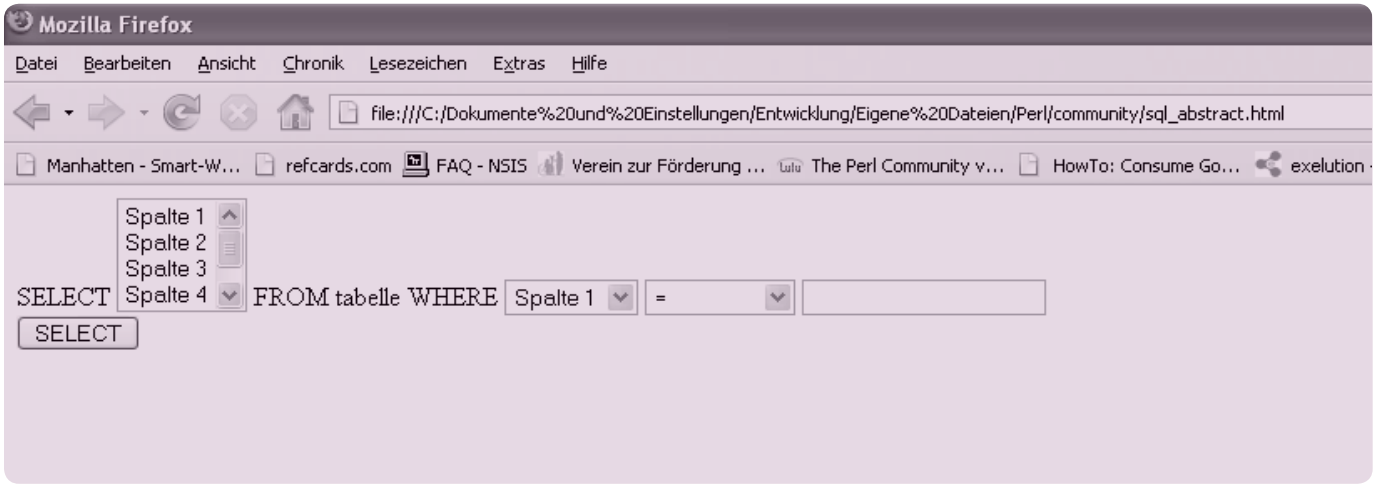
Mit Hashreferenzen werden UND-Verknüpfungen in der `WHERE`-Bedingung dargestellt (Listing 4), mit Arrayreferenzen ODER-Verknüpfungen (Listing 5).

Diese Datenstruktur kann beliebig tief strukturiert sein, damit alle möglichen `WHERE`-Bedingungen dargestellt werden können. Ein `WHERE ID = ? AND (col5 = ? OR col7 = ?)` wird zum Beispiel mit

```
my $where = {
    -and =>
      [
        ID => { '!=' => 129 },
        [
          col5 => 3,
          col7 => 19,
        ],
      ]
 };
```

Listing 6

erreicht. Gerade wenn man noch nicht viel mit Modulen wie `DBIx::Class` gearbeitet hat, ist die Formulierung der Bedingung sehr kompliziert und bedeutet häufig "Rumprobierei".



Wird ein einfacher String oder eine Zahl für die Spalte angegeben, wird Standardmäßig das '=' als Vergleichsoperator genommen. Soll ein anderer Vergleich genommen werden, muss dies in einer Hashreferenz angegeben werden (Listing 7).

```
my $where = [
   ID => { '!=' => 129 },
   col5 => 3,
];
```

Listing 7

Mit dem Beispiel sieht die Bedingung so aus:

```
WHERE ( ( ID != ? ) OR ( col5 = ? ) )
```

Es gibt Fälle, in denen Funktionen des Datenbanksystems genutzt werden sollen. Die kennt `SQL::Abstract` natürlich nicht alle. Um diese dennoch nutzen zu können, muss eine Referenz auf einen String geliefert werden. Das Beispiel in Listing 8 verwendet die `REGEXP`-Funktion von MySQL (`WHERE ( REGEXP(test) )`).

```
my $where = {
   '' => \'REGEXP(test)',
};
```

Listing 8

Das Modul bietet keine großen Vorteile, wenn die zu selektierenden Spalten schon von vornherein feststehen. Aber es bietet dann einen großen Vorteil, wenn die Spalten erst im Programmablauf festgelegt werden (können). Dann muss man nicht mit eigenen `map`- oder `grep`-Lösungen arbeiten, wenn man die ?-Notation von DBI verwenden will, da das dann alles von `SQL::Abstract` übernommen wird.

Ein Manko bei dem Modul ist, dass Tabellen- und Spaltennamen nicht gequotet werden. So treten bei Spaltennamen mit Leerzeichen Probleme auf.

Ein Beispiel für den Einsatz von `SQL::Abstract(::Limit)?` ist zum Beispiel die Suche in der Datenbank über ein Webformular. Ein User kann sich dabei aussuchen, welche Spalten selektiert werden sollen - und mit welchen Bedingungen. In Abbildung 1 ist ein Screenshot von so einem Formular zu sehen. Listing 9 zeigt das Skript, das dieses Formular auswertet und dann die Ergebnisse als einfache Tabelle ausgibt.

Renée Bäcker

```
#!/usr/bin/perl

use strict;
use warnings;
use CGI;
use CGI::Carp qw(fatalsToBrowser);
use DBI;

use lib qw(./lib);
use SQL::Abstract::Limit;

my $cgi = CGI->new;
print $cgi->header;

my %params = $cgi->Vars;

##
# der nachfolgende Code ist relevant für SQL::Abstract

my $sal   = SQL::Abstract::Limit->new( limit_dialect => 'LimitXY' );
my $where;

$where = {
     $params{wherecol} => { $params{whereop} => $params{whereval} },
} if $params{whereval};

my $limit = defined $params{limitval} ? $params{limitval} : undef;

my ($sql,@binds) = $sal->select(
  'testtabelle', [$cgi->param('columns')], $where, 'col1', $limit
);

# der relevante Code ist zu Ende
##

my $dbh = DBI->connect( "DBI:mysql:xxx:xxx","xxx","xxx" ) or die $DBI::errstr;
my $sth = $dbh->prepare( $sql ) or die $dbh->errstr;
$sth->execute( @binds )         or die $dbh->errstr;

print "<table>";
while( my @row = $sth->fetchrow_array ){
   print "<tr>",
         map{ "<td>$_</td>" }@row,
         "</tr>";
}
print "</table>";
```

Listing 9