

Perl und Datenbanken (am Beispiel MySQL)

Einführung

Durch MySQL haben Datenbanken einen wahren Hype erlebt. Das große Schlagwort der vergangenen Jahre hieß „dynamische Webseiten“. Bei den meisten dynamischen Webseiten läuft ein Datenbanksystem im Hintergrund. Um diese Daten aus der Datenbank anzeigen zu können ist der Zugriff auf die Datenbank notwendig.

Die Schnittstellen zu einer Datenbank sind definiert. Dennoch ist es umständlich alles selbst zu programmieren. Aus diesem Grund ist es sinnvoll Module von CPAN zu verwenden.

Ein Datenbanksystem ist das ganze System, wie z.B. Oracle oder MySQL. Eine Datenbank ist eine Sammlung von Daten. Diese Daten sind in Tabellen organisiert.

Für unsere Beispiele nehmen wir eine einfache Datenbank, z.B. ein Gästebuch mit nur einer Tabelle, in der alle Einträge gespeichert sind. Wenn Sie diese Datenbank nachmachen wollen, legen Sie eine Datenbank über phpMyAdmin oder über das Kommandozeilentool an. Danach Erstellen Sie die Tabelle. Hierfür finden Sie am Ende die SQL-Befehle.

Die Tabelle sieht folgendermaßen aus:

Field	Type	Null	Key	Default	Extra
ID	int(11)		PRI	0	
Name	varchar(100)				
Homepage	varchar(100)	YES		NULL	
Ort	varchar(50)	YES		NULL	
EMail	varchar(100)				
Nachricht	longtext				
date	varchar(10)				

Für die hier gezeigten Codebeispiele verwenden wir das DBI-Modul.

Kein Anschluss unter dieser Nummer

Um Abfragen durchführen zu können, muss erst einmal eine Verbindung zu der Datenbank hergestellt werden. Dazu benötigt man Datenbankname, Username, Userpasswort sowie den Hostnamen. Der Hostname ist meistens `localhost` bzw. `127.0.0.1`. Für den Verbindungsaufbau gibt es die Methode `connect()`. Als Rückgabewert bekommt man den sogenannten Datenbankhandler.

```
#!/usr/bin/perl

use strict;
use warnings;
use DBI;

my $user    = 'username';
my $pass    = 'userpasswort';
my $db      = 'datenbankname';
my $host    = 'localhost';
my $driver  = "DBI:mysql:$db:$host";

my $dbh     = DBI->connect($driver,$user,$pass) or die $DBI::errstr;
```

Wenn jetzt keine Fehlermeldung kommt, ist die Verbindung zur Datenbank hergestellt.

Schau mal, wer sich alles eingetragen hat.

Jetzt können wir uns auf die Suche nach den gewünschten Daten machen. Bei größeren Datenbanken sollte man sich schon ein wenig mit SQL – der Abfragesprache für Datenbanken – auskennen. Doch für das kleine Gästebuch reichen einfache SQL-Abfragen. Wir wollen einfach alles wissen. Somit lautet der SQL-Befehl:

```
SELECT * FROM guestbook;
```

Doch wie bekommen wir jetzt die Daten?

Dazu müssen wir die Abfrage erst vorbereiten – mit der entsprechenden Methode `prepare()`. Als Rückgabewert bekommt man den Statementhandler.

```
my $statement = 'SELECT * FROM guestbook';
my $sth = $dbh->prepare($statement) or die $DBI::errstr;
```

Jetzt ist die Abfrage vorbereitet, aber Ergebnisse haben wir immer noch nicht. Dazu muss die Abfrage erstmal ausgeführt werden. Bis jetzt ist sie vorbereitet, aber noch nicht ausgeführt. Warum diese Teilung in zwei Schritte sinnvoll ist, zeige ich später. Doch erstmal wollen wir die Abfrage ausführen – mit `execute()`.

```
$sth->execute() or die $DBI::errstr;
```

Bei der Abfrage kommen jetzt die Spalten in der Reihenfolge, wie sie bei der Erstellung der Tabelle angegeben wurden. Die Reihenfolge ist an der obigen Grafik zu erkennen.

Diese müssen wir Zeilenweise „abholen“ - das sogenannte fetchen. Wie gesagt, werden die Ergebnisse Zeilenweise geliefert, so dass wir – um alle Ergebnisse zu bekommen – eine Schleife einsetzen müssen. Mit der Methode `fetchrow_array()` holen wir ein Array, in dem eine Zeile enthalten ist. Ein Element entspricht einer Spalte.

```
my @ergebnisse;
while(my @row = $sth->fetchrow_array()){
    push(@ergebnisse,\@row);
}
```

Jetzt haben wir alle Ergebnisse im Array `@ergebnisse`.

Darstellung ohne HTML

Um uns das HTML im Code zu sparen, verwenden wir zur Darstellung der Ergebnisse ein Template und das Modul `HTML::Template`, dessen Verwendung ich schon mal unter http://perl.renee-baecker.de/HTML_Template.pdf beschrieben habe. Wie das Template aussieht ist am Ende zu finden.

Um es nicht zu kompliziert zu machen, geben wir alle Einträge am Stück aus, ohne den Schnickschnack wie „10 Einträge pro Seite“ usw.

Wir erzeugen ein Objekt von `HTML::Template` und füllen die Tabelle mit unseren Ergebnissen.

```
use HTML::Template;
my $tpl_file = '/path/to/template.tpl';
my $template = HTML::Template->new(filename => $tpl_file);
my @eintraege;
foreach my $eintrag(@ergebnisse){
    my %hash = (ID => $eintrag->[0],
               NAME => $eintrag->[1],
               HP => $eintrag->[2],
               ORT => $eintrag->[3],
               MAIL => $eintrag->[4],
               TEXT => $eintrag->[5],
               DATE => $eintrag->[6],);
    push(@eintraege,\%hash);
}
$template->param(EINTRAEGE => \@eintraege);
print $template->output();
```

So können jedoch nur schon existierende Einträge ausgegeben werden. Um Einträge zu bekommen muss man wie folgt vorgehen:

- *) Formular basteln
- *) CGI-Skript zum Auswerten des Formulars
 - Datenbankverbindung herstellen
 - mögliche Fehler abfangen
 - INSERT-Statement vorbereiten und ausführen

Viel Spaß beim Ausprobieren.

Anhang

Skript komplett:

```
#!/usr/bin/perl

use strict;
use warnings;
use DBI;
use HTML::Template;

print "Content-type: text/html\n\n";

my $tpl_file = '/path/to/template.tpl';
my $template = HTML::Template->new(filename => $tpl_file);
my @eintraege;

my $user     = 'username';
my $pass     = 'userpassword';
my $db       = 'datenbankname';
my $host     = 'localhost';
my $driver   = "DBI:mysql:$db:$host";

my $dbh      = DBI->connect($driver,$user,$pass) or die $DBI::errstr;
my $sql      = 'SELECT * FROM guestbook';
my $sth      = $dbh->prepare($sql) or die $DBI::errstr;

$sth->execute() or die $DBI::errstr;

my @ergebnisse;
while(my @row = $sth->fetchrow_array()){
    push(@ergebnisse, \@row);
}

foreach my $eintrag(@ergebnisse){
    my %hash = (ID => $eintrag->[0],
                NAME => $eintrag->[1],
                HP => $eintrag->[2],
                ORT => $eintrag->[3],
                MAIL => $eintrag->[4],
                TEXT => $eintrag->[5],
                DATE => $eintrag->[6],);
    push(@eintraege, \%hash);
}

$template->param(EINTRAEGE => \@eintraege);
print $template->output();
```

SQL-Befehle für die Tabelle guestbook:

```
CREATE TABLE `guestbook` (
  `ID` int(11) NOT NULL default '0',
  `Name` varchar(100) NOT NULL default '',
  `Homepage` varchar(100) default NULL,
  `Ort` varchar(50) default NULL,
  `EMail` varchar(100) NOT NULL default '',
  `Nachricht` longtext NOT NULL,
  `date` varchar(10) NOT NULL default '',
  PRIMARY KEY (`ID`)
) TYPE=MyISAM;
```

HTML-Template:

```
<html>
  <body>
    <table>
      <!-- TMPL_LOOP NAME=EINTRAEGE -->
      <tr><td>Eintrag Nr: <!-- TMPL_VAR NAME=ID -->
          vom <!-- TMPL_VAR NAME=DATE --></td></tr>
      <tr><td><!-- TMPL_VAR NAME=NAME -->
          aus <!-- TMPL_VAR NAME=ORT -->
          (Mail: <!-- TMPL_VAR NAME=MAIL -->
           HP: <!-- TMPL_VAR NAME=HP -->)
        </td></tr>
      <tr><td><!-- TMPL_VAR NAME=TEXT --></td></tr>
    <!-- /TMPL_LOOP --></table>
  </body>
</html>
```

Links für Perlprogrammierer:

<http://www.perl.com> – Perl

<http://www.perl-community.de> – das große deutschsprachige Perl-Forum

<http://www.perlmonks.org> – englischsprachiges Perl-Forum

<http://www.cpan.org> – Archiv mit Perl-Modulen

<http://www.perldoc.com> – Online-Version der perldocs (Dokumentation)

<http://www.activestate.com> – Perl für Windows

<http://www.oreilly.com> – Perl-Bücher

<http://regenechsen.de> – Tutorial zu Regulären Ausdrücken

Der Autor:

Renee Bäcker ist Gesellschafter der Smart-Websolutions Windolph, Morgenroth und Bäcker GbR.

Er programmiert seit ein paar Jahren Perl und ist Moderator im Perl-Community-Forum. Er ist hauptsächlich im Bereich der Bioinformatik und auch im Webbereich tätig. (<http://www.renee-baecker.de>) < perl@renee-baecker.de >