TRILOS

„Containerfrachter“

Komplexe Datenstrukturen mit Objekten managen

Von Ulrich Hacke

PHP Usergroup Hannover – 12.06.2003

Trilos GbR
Am Rathaus 15
30592 Ronnenberg

1 Einführung

Dynamisch erzeugte Webseiten sind das tägliche Brot des PHP-Programmierers. Wann immer jedwede Form von Content ausgegeben werden soll, müssen PHP-Funktionen geschrieben und aufgerufen werden, die anhand von Suchkriterien die gewünschten Elemente aus einer Datenbank holen und ihn anschließend im spezifischen Layout zurückschreiben.

Schnell befindet man sich in der Situation wieder, dass der gleiche Content immer wieder in unterschiedlichem Layout dargestellt werden muss, z.B. als Navigation, als Vollansicht oder als Liste („Siehe auch..."). Natürlich kann man diese Problematik dadurch lösen, indem man an jeder benötigten Stelle im Quellcode einen passenden PHP-Block schreibt, der die Daten liest und ausgibt. Das würden Anfänger tun – Fortgeschrittene würden die Blöcke bereits in einer Funktion kapseln... und Profis? Profis arbeiten mit Objekten. Objekte (oder Klassen) bieten viele Vorteile:

- Wiederverwendbarkeit (code once use often)
- Abstraktion
- Kapselung von Funktionalität an zentraler Stelle
- Flexibilität

Ein Vergleich aus der materiellen Welt macht's deutlich: möchte man viele unterschiedliche Waren um die halbe Welt verschicken, ist es am einfachsten, sie alle in einen Container zu stopfen und diesen auf ein Schiff zu verladen. Das ist einfach und geht schnell. Am Bestimmungsort werden alle Waren wieder entladen und stehen zur Verfügung. Containerfrachter sind aus dem heutigen Güterverkehr nicht mehr wegzudenken. Und mit PHP und ein wenig Know-how können wir das auch. Wie es funktioniert, erklärt dieses Tutorial.

2 Anwendungsbeispiel

Stellen wir uns einen Newsserver vor, der aktuelle Nachrichten aus verschiedenen Themengebieten bereithält. Eine Nachricht besteht aus ihrer Überschrift, einem Anreißer (auch „Teaser" genannt), dem Veröffentlichungsdatum und natürlich dem Nachrichtentext selbst. Jede Nachricht ist genau einem Thema (Rubrik) zugeordnet.

Unser PHP-Code soll folgendes leisten:

1. Übersicht aller Rubriken
2. Auflisten von News nach optionalen Suchkriterien
3. Vollansicht einer News

Im Downloadbereich auf www.hacke.net/php steht eine mySQL-Datenbank mit Beispielnews bereit. Die eingesetzten PHP-Skripte sind ebenfalls dort verfügbar. Der Aufbau der verwendeten Datenbank ist in Kapitel 6 nachzulesen.

3 Herangehensweise

Wie besprochen, wollen wir die Programmieraufgabe mit Hilfe von Objekten lösen. Alle Content-Elemente sollen in einem Datencontainer gelagert und erst bei Benutzung (Darstellung) „ausgepackt" werden.

Dazu benötigen wir zwei Objekte: ein Objekt „Rubrik" und ein Objekt „Nachricht". Kurz zur Erinnerung: Objekte sind besondere Datenstrukturen; sie besitzen Eigenschaften und Methoden. Würden wir beispielsweise ein Auto als Objekt ansehen, könnte eine Eigenschaft seine Farbe (rot) und eine Methode „Beschleunigen" sein. Merke: Eigenschaften können gelesen oder geschrieben werden, Methoden sind einzelne Code-Blöcke, die ir-

gendeine Funktionalität haben und Eigenschaften verändern können. Methoden können Parameter übergeben bekommen oder Daten zurückliefern.

In PHP werden Objekte als Klassen über das Sprachkonstrukt class definiert; ihre Eigenschaften werden mit var eingeleitet. Methoden werden wie gewöhnliche Funktionen innerhalb des Klassenblocks geschrieben. Diejenige Methode, die den gleichen Namen wie die Klasse trägt, wird als Konstruktor bezeichnet und wird automatisch aufgerufen, wenn wir uns eine Instanz der Klasse erzeugen.

3.1 Container-Konstruktion

Wir definieren uns nun die nötigen Objekte für unser Beispiel:

```
// Objekt "Rubrik"
class Rubrik {
    // Eigenschaften
    var $rubrik_id;
    var $bezeichnung;

    // Konstruktor
    function Rubrik() {
    }

    // Daten lesen
    function Lesen() {
    }
}

// Objekt "Nachricht"
class Nachricht {
    // Eigenschaften
    var $artikel_id;
    var $datum_uhrzeit;
    var $ueberschrift;
    var $anreisser;
    var $volltext;
    var $rubrik_id;
    var $rubrik;
    var $suchbegriff;
    var $_anzahl;

    // Konstruktor
    function Rubrik() {
    }

    // Daten lesen
    function Lesen() {
    }
```

Die Eigenschaften beider Objekte bieten den Platz für alle Datenfelder einer Rubrik bzw. einer Nachricht. Jedes Objekt besitzt neben seinem Konstruktor eine Methode Lesen(), die später mit der Datenbank kommunizieren und die Daten entsprechend aufbereiten wird.

3.2 Erster Container: Rubriken

Wenden wir uns als erstes den Rubriken zu – gewünscht war eine Liste aller verfügbaren Rubriken. Da wir nicht wissen, wie viele Rubriken es gibt – Onlineredakteure reden nicht mit Programmierern – können wir im Vorfeld also keine festgelegte Zahl von Rubrik-Objekten instanzieren. Stattdessen lassen wir uns eine Liste der Rubriken geben: die Lesen()-Methode soll daher ein Array von Rubrik-Objekten lesen.

Als erstes brauchen wir den Konstruktor. Seine Aufgabe besteht darin, die übermittelten Daten als Objekteigenschaften zu speichern:

```
// Konstruktor
function Rubrik($rubrik_id, $bezeichnung) {
    $this->rubrik_id = $rubrik_id;
    $this->bezeichnung = $bezeichnung;
}
```

Merke: alle Variablen innerhalb einer Klassenmethode sind lokal – wie bei herkömmlichen Funktionen auch. Möchte man auf die Eigenschaften einer Klasse zugreifen, muss der Klassenname vorangestellt werden. Handelt es sich um die „eigene" Klasse, wird $this verwendet. Der Pfeiloperator trennt den Klassennamen von der Eigenschaft (oder der Methode) ab. Letztere darf übrigens auf keinen Fall mit einem $ beginnen, sonst wird sie als variable Variable interpretiert – was in einem anderen Fall durchaus sinnvoll einzusetzen ist, führt in den meisten Fällen aber nur zu schwer nachvollziehbaren Fehlern. Aber dazu später mehr.

Der zugehörige Code für Lesen() innerhalb von Rubrik sieht so aus:

```
// Rubrikdaten lesen
function Lesen() {
    global $dbh;

    $ret_val = array();

    // SQL an Datenbank senden
    $query = mysql_query("select rubrik_id, bezeichnung from rubrik order by bezeichnung asc", $dbh);
    // Ergebnis in Objekt wandeln...
    while($row = mysql_fetch_array($query)) $ret_val[] = new Rubrik($row["rubrik_id"], $row["bezeichnung"]);

    // ...und zuruecckliefern
    return $ret_val;
}
```

Da wir eine ganze Liste von Objekten geliefert haben wollen, erstellen wir uns zunächst ein leeres Array ($ret_val). Anschließend wird die Datenbank befragt und das gewonnene Recordset in ein Objekt umgewandelt: unser $ret_val wird mit immer neuen Instanzen des Rubrik-Objektes befüllt und zum Schluss via return an den aufrufenden Code geliefert. Sollten keine Rubriken vorhanden gewesen sein (weil der Online-Redakteur gerade Urlaub hatte), dann bekommen wir ein leeres Array zurück. Noch praktischer wäre natürlich, in diesem Fall ein false zurück zu liefern. Damit könnte zusätzlich geprüft werden, ob tatsächlich eine Liste empfangen wurde:

```
If($rubriken = Rubrik::Lesen()) {
    ..
}
```

Mit dem folgenden Code lesen wir die Rubrikdaten als Objektliste und geben sie anschließend aus:

```
$rubriken = Rubrik::Lesen();
foreach($rubriken as $rubrik) {
    echo $rubrik->bezeichnung . "\n";
}
```

Moment! Hieß es nicht, man müsse den Pfeiloperator verwenden, um auf Methoden oder Eigenschaften einer Klasse zuzugreifen? Im Prinzip ja; allerdings wurde der ::-Operator im Beispiel aus gutem Grund verwendet, denn es existiert ja zum Zeitpunkt des Aufrufs noch gar kein Objekt. Via :: können wir jedoch auf Klassenmethoden zugreifen, von deren Klasse es keine Instanz gibt. Ganz wichtig: da es noch kein Objekt gibt, dürfen auch keine Objektvariablen oder der Ausdruck $this verwendet werden. Sollten dennoch Objektvariablen gesetzt werden, gehen ihre Inhalte beim Rücksprung aus der Klassenmethode unweigerlich verloren.

Der Aufruf von Lesen() liefert und das gewünschte Array mit den Rubrikinhalten zurück, die wir in einer foreach-Schleife ausgeben. Et voilá: Aufgabe 1 wurde gelöst.

3.3 Zweiter Container: News

Nachdem wir das grundlegende Know-how erlernt haben, gehen wir bei den News ganz ähnlich vor: auch hier verwenden wir eine Methode Lesen(), um eine Liste aller derjenigen Nachrichten zu erhalten, die auf unsere Suchkriterien passen. Diese übermitteln wir natürlich ebenfalls in Objektform: dazu wird ein „Muster" vom Objekttyp Nachricht instanziert und mit denjenigen Eigenschaften beschrieben, welche die zu liefernden Objekte erfüllen müssen. Mit diesem Trick können wir Aufgabe 2 und drei gleich auf einen Streich lösen und gewissermaßen „zwei Fliegen mit einer Klasse" erschlagen. Die Eigenschaft $_anzahl (zur besseren Unterscheidung mit einem Unterstrich eingeleitet) soll die Anzahl der passenden Nachrichten liefern:

```
// Vorlage erstellen
$vorlage = new Nachricht();
// Suchkriterium setzen
$vorlage->suchbegriff = "ferrari";
// ggf. weitere Kriterien
...

$nachrichten = Nachricht::Lesen(&$vorlage);
echo $vorlage->_anzahl . " Treffer<br><br>";
foreach($nachrichten as $nachricht) {
    echo $nachricht->ueberschrift . "<br>;
}
```

Als erstes instanzieren wir eine $vorlage und weisen der Eigenschaft $suchbegriff einen Wert zu. Diese $vorlage übergeben wir der Lesen-Methode der Nachrichtenklasse als Parameter. Auffällig ist dabei das „kaufmännische Und". Es weist den PHP-Interpreter an, das Originalobjekt $vorlage als Parameter zu übergeben (call by reference). PHP verwendet bei der Parameterübergabe jedoch grundsätzlich einen call by value, d.h. es wird die Kopie des Parameters übergeben. Wird innerhalb der Funktion oder der Klassenmethode der Wert des Parameters verändert, hat das nur Auswirkungen innerhalb der Funktion – Parameter und Funktionsvariable sind grundsätzlich lokal.

Innerhalb der Lesen()-Methode müssen wir die zuvor gesetzten Suchkriterien auswerten und in das SQL-Statement integrieren, welches die passenden Datensätze aus der Datenbank liest. Um die Anzahl der benötigten IF-Konstrukte (und die der Codezeilen) möglichst klein zu halten, verwenden wir ein Array namens $params, dem wir alle Such-

kriterien in vorgefertigter SQL-Syntax zuordnen (so sie gesetzt wurden). Vorher prüfen wir, ob überhaupt eine Vorlage übergeben wurde:

```
function Lesen($muster="") {
    global $dbh;

    // Grundlegendes SQL-Statement
    $sql = "select n.*, r.rubrik_id, r.bezeichnung from nachricht n, rubrik r
where n.rubrik_id = r.rubrik_id";

    // "Muster-Objekt" erhalten?
    if(is_object($muster)) {
        $params = array();
        // Suchkriterien in Array speichern...
        if($muster->rubrik_id!="") $params[] = "n.rubrik_id = " . $muster-
>rubrik_id;
        if($muster->artikel_id!="") $params[] = "n.artikel_id = " . $muster-
>artikel_id;
        if($muster->suchbegriff!="") $params[] = "(n.ueberschrift like '%" .
addslashes($muster->suchbegriff) . "%' or n.anreisser like '%" .
addslashes($muster->suchbegriff) . "%' or n.volltext like '%" . addslashes($muster-
>suchbegriff) . "%')";

        // ...und ggf. an SQL-Statement anhaengen
        if(count($params)>0) $sql .= " and " . implode(" and ", $params);
    }
    // SQL-Statement abschliessen und an DB senden
    $sql .= " order by n.rubrik_id, n.zeitstempel desc";
    $query = mysql_query($sql, $dbh);

    // Anzahl der Ergebnisse merken
    $muster->_anzahl = mysql_affected_rows($dbh);
    // Ergebnisdaten in Objekt wandeln...
    while($row = mysql_fetch_array($query, MYSQL_ASSOC)) $ret_val[] = new
Nachricht($row);

    // ...und zurueckliefern
    return $ret_val;
}
```

Wie schon bei den Rubriken erstellen wir ein Rückgabe-Array von passenden Nachrichten-Objekten. Da es hier diesmal eine ganze Reihe von Eigenschaften zu übergeben gilt und wir natürlich unnötige Tipparbeit sparen wollen, übergeben wir diesmal gleich einen kompletten Datensatz als assoziatives Array und lassen den Konstruktor die eigentliche Arbeit übernehmen:

```
// Konstruktor
function Nachricht($row="") {
    // Ergebnis-Array erhalten?
    if(is_array($row)) {
        // Alle Felder durchgehen und Klasseneigenschaften zuweisen
        while(list($key,$value) = each($row)) $this->$key = $value;
    }
}
```

An dieser Stelle setzen wir bei der Eigenschaftenzuweisung ganz einfach variable Variablen ein – denn das übergebene assoziative Array bringt die Namen der Eigenschaften ja passenderweise als Key gleich mit.

Mit diesem Code haben wir Aufgabe 2 gelöst; es müssen nur die Suchkriterien in der Vorlage richtig gesetzt werden. Sollten keinerlei Kriterien gesetzt werden, erhalten wir alle Nachrichten. Das gleiche gilt für Aufgabe 3 – schließlich ist die Eigenschaft $nachricht_id ja ebenfalls ein Suchkriterium.

3.4 Performance

Man könnte einwenden, dass der Code insofern nicht performant ist, als dass er auch solche Eigenschaftsdaten liefert, die möglicherweise gar nicht verwendet werden. In der Regel ist das aber zu vernachlässigen, da PHP die Daten eh im Cache-RAM zwischenpuffert. Bei highload-Sites dagegen sollte man sich dagegen vielmehr fragen, inwieweit PHP das richtige Instrument dafür ist.

3.5 Ausblick und Verbesserung

Die Nachrichtenklasse könnte um einige Zusatzfeatures erweitert werden:

- **Sortierung**
  Neben den Suchkriterien wären auch ein oder mehrere Sortierkriterien anzugeben – im derzeitigen Stadium wird (festverdrahtet) nach dem Feld „datum" sortiert.
- **Ergebnissteuerung**
  Des weiteren könnte ein „Limit"-Wert gesetzt werden, d.h. es werden maximal so viele Nachrichten zurückgeliefert, wie in $limit angegeben sind.

Weitere Verbesserungsvorschläge inklusive Beispiel bringt das Kapitel „Laderampe" (siehe 5).

4 Tipps zum Debuggen

Bei der Arbeit mit Klassenobjekten leistet die PHP-Funktion print_r() hervorragende Dienste. Sie zeigt Informationen über eine Variable oder einen Ausdruck in lesbarer Form an. Einfache Werte werden direkt ausgegeben, Arrays und Objekte werden dagegen als eingerückte Schlüssel / Wert-Paare ausgegeben. Ein Beispiel für unser Rubriken-Objekt:

```
$rubriken = Rubrik::Lesen();
print_r($rubriken);
```

ergibt

```
Array
(
    [0] => rubrik Object
        (
            [rubrik_id] => 1
            [bezeichnung] => Lifestyle
        )

    [1] => rubrik Object
        (
            [rubrik_id] => 2
            [bezeichnung] => Politik
        )
    )
    usw.
)
```

Vor der Verwendung von print_r() sollte man jedoch das HTML-Tag <pre></pre> einsetzen, da print_r() neue Zeilen nur mit einem CRLF, nicht jedoch mit einem
 einleitet. Alternativ kann man sich das Ergebnis natürlich auch im Quelltext anstatt im Browser ansehen...

5 Erweiterung: Die „Laderampe"

Nichts ist bekanntlich so gut, als dass man es nicht noch verbessern könnte. Mit unserem PHP-Code haben wir nun einen Frachter, der den Content transportieren und verwalten kann. Mit Hilfe einer „Laderampe" könnten wir es einem Online-Redakteur ermöglichen, über ein HTML-Formular Content „einzuladen" (Neueingabe) oder zu verändern (Bearbeiten, Löschen). Dazu erweitern wir unser Objekt Nachricht um die folgenden Methoden:

```
function Formular($id="") {}
function Speichern($id="") {}
function Loeschen($id) {}
```

Die Erzeugung des HTML-Formulars soll das Objekt natürlich ebenfalls übernehmen. Übrigens: die hier beschriebene Vorgehensweise ist im Prinzip mit dem bekannten „phpMyAdmin" vergleichbar.

5.1 Formularerzeugung

Der optionale Parameter der Methode Formular() gibt an, ob ein bestehender Datensatz bearbeitet (dann muss $id einen vorhandenen Wert besitzen) oder ein neuer angelegt werden soll. Die Methode liest daraufhin die Tabellenstruktur aus, in welcher der Datensatz gespeichert werden soll und erstellt ein dafür passendes HTML-Formular. Die einzelnen Formularelemente werden nach ihren Datenbankfeldern benannt und – falls eine $id übergeben wurde – mit dem derzeitigen Inhalt befüllt:

```
// HTML-Formular erstellen
function Formular($id="") {
    global $dbh;

    // ggfs. Nachricht lesen
    if($id!="") {
          $query = mysql_query("select * from nachricht where nachricht_id =
$id", $dbh);
          $data = mysql_fetch_array($query);
    }

    // Tabellenstruktur ermitteln
    $query = mysql_query("show fields from nachricht", $dbh);
    while($row = mysql_fetch_array($query)) {
          echo ucfirst($row["Field"]) . "<br>";
          if(ereg("text", $row["Type"])) {
                echo "<textarea name='".$row["Field"]."' cols='30'
rows='8'>".stripslashes($data[$row["Field"]])."</textarea><br>";
          } else {
                echo "<input type='text' name='".$row["Field"]."' size='30'
value='".stripslashes($data[$row["Field"]])."'><br>";
          }
    }
}
```

Den Namen des Datenbankfeldes benutzen wir übrigens auch gleich als Label für das jeweilige Eingabefeld. Zusätzlich unterscheiden wir nach dem Typ der Datenbankfelder: handelt es sich um ein Feld vom Typ Text (tinytext, mediumtext, text), stellen wir eine <textarea> dar, in allen anderen Fällen ein einzeiliges <input>-Feld.

Wurde eine $id übergeben, dann erhalten alle Eingabefelder als Vorbelegung (value-Attribut) den aktuell gespeicherten Wert aus dem assoziativen Array $data. Das alles können wir bequem in einer einzigen while-Schleife erledigen, da die Feldnahmen und die Namen der Eingabefelder identisch sind und wir auf die Werte über das $data-Array zugreifen können.

5.2 Daten speichern

Auch hier wird anhand einer eventuell übergebenen $id entscheiden, ob wir ein INSERT oder ein UPDATE auf die Datenbank fahren. In jedem Fall wird auch hier die Struktur derjenigen Tabelle benötigt, in der wir den Datensatz speichern wollen. Über das assoziative Array $http_POST_VARS können wir die Formulardaten innerhalb unsere Methode Speichern() auswerten.

```
   // Nachricht speichern
   function Speichern($id="") {
       global $dbh;
       global $HTTP_POST_VARS;

       // Tabellenstruktur ermitteln
       $query = mysql_query("show fields from nachricht", $dbh);
       while($row = mysql_fetch_array($query)) {
              $felder[] = $row["Field"];
              $werte[] = "'".addslashes($HTTP_POST_VARS[$row["Field"]])."'";
              $update[] =
$row["Field"]."='".addslashes($HTTP_POST_VARS[$row["Field"]])."'";
       }
       if($id=="") {
              // Neuen Datensatz speichern
              echo "insert into nachricht (".implode(",", $felder).") values
(".implode(",", $werte).")";
       } else {
              // Bestehenden Datensatz updaten
              mysql_query("update nachricht set ".implode(",", $update)." where
nachricht_id = $id", $dbh);
       }
   }
```

Je nach „Auftragslage" – also INSERT oder UPDATE – bauen wir uns das jeweils passende SQL-Statement zusammen. Dabei vereinfachen wir, indem wir die Feldnamen und –werte zunächst in Arrays halten und sie erst im SQL-Statement via implode() in eine Zeichenkette umwandeln. Unsere Arbeit wird dadurch erleichtert, dass mySQL es zulässt, dass jeder Datentyp in (einfachen) Anführungszeichen übergeben werden kann, auch wenn es sich um numerische Daten handelt.

Die Methode Speichern() sollte auf jeden Fall zur Sicherheit noch mit Plausibilitätsprüfungen ausgestattet werden. Zusätzlich könnten einige Datenfelder Pflichteingaben sein, während andere optional wären. In diesem Fall sollte Speichern() erst dann aktiv werden, wenn auch wirklich alle benötigten Felder einen gültigen Inhalt aufweisen.

5.3 Daten löschen

Die Methode Loeschen() ist kurz und knapp gehalten. Als Parameter erwartet sie die $id des zu löschenden Datensatzes und fährt anschließend ein DELETE auf die Datenbank. Auch hier sollten natürlich Sicherheitsprüfungen vorgeschaltet werden, bevor tatsächlich gelöscht wird.

```
// Nachricht loeschen
function Loeschen($id) {
    global $dbh;

    mysql_query("delete from nachricht where nachricht_id = $id", $dbh);
}
```

5.4 Anwendung der "Laderampe"

Die erweiterten Funktionen (Methoden) der „Laderampe" innerhalb des Objektes Nachricht bringen selbstverständlich nur die reine Funktionalität mit. Angesteuert werden müssen sie natürlich nach wie vor über ein PHP/HTML-Skript. Dank der objektorientierten Programmierung ist der Quellcode der Steuerskripte aber erfreulich kurz.

Neue Nachricht:

```
<form action="neu.php" method="post">
<?
$n = new Nachricht();
if($_action=="go") $n->Speichern();
$n->Formular();
?>
<input type="submit" value="Speichern">
<input type="hidden" name="_action" value="go">
</form>
```

Das Bearbeiten einer bestehenden Nachricht erfolgt analog – hier muss in den Methoden Speichern() und Formular lediglich die $id angegeben werden.

5.5 Noch ein Ausblick – auf die „Laderampe"

Wir haben gelernt, mit Hilfe von Objekten schnell und einfach Content in einer Datenbank zu verwalten. Die vorgestellte Herangehensweise zeichnet sich vor allem durch ihre Flexibilität aus: sollten beispielsweise neue Datenbankfelder für das Objekt „Nachricht" hinzukommen, muss der Quellcode zur Pflege nicht einmal angefasst werden; schließlich sucht er sich seine Informationen selbst zusammen. Lediglich auf der Ausgabeseite müssen Veränderungen vorgenommen werden. Was könnte die Laderampe noch alles leisten?

- Plausibilitäts- und Vollständigkeitsprüfungen
- Unterscheidung zwischen Pflicht- und optionalen Feldern
- Unterstützung spezieller Datentypen, z.B. Datums- und Zeitangaben
- Datei-Uploads, Check- und Radioboxen

Man sieht einmal mehr, dass mit dem richtigen Know-how und ein bisschen PHP fast nichts unmöglich ist. Mit Hilfe des „Containerfrachters" und seiner „Laderampen" lassen sich kleine CMS-Systeme im Handumdrehen erstellen und größere Sites wesentlich einfacher und effizienter realisieren.

6 Datenbank

Aufbau der mySQL-Datenbank für das im Tutorial verwendete Beispiel:

nachricht					
Feldname	**Feldtyp**	**Null**	**Key**	**Standard**	**Extra**
nachricht_id	int(11)		PRI		auto_increment
rubrik_id	int(11)			0	
zeitstempel	int(11)			0	
ueberschrift	varchar(250)				
anreisser	mediumtext				
volltext	text				

rubrik					
Feldname	**Feldtyp**	**Null**	**Key**	**Standard**	**Extra**
rubrik_id	int(11)		PRI		auto_increment
bezeichnung	char(100)			0	

```
CREATE TABLE rubrik (
   rubrik_id int(11) NOT NULL auto_increment,
   bezeichnung char(100) DEFAULT '0' NOT NULL,
   PRIMARY KEY (rubrik_id)
);

CREATE TABLE nachricht (
   nachricht_id int(11) NOT NULL auto_increment,
   rubrik_id int(11) DEFAULT '0' NOT NULL,
   zeitstempel int(11) DEFAULT '0' NOT NULL,
   ueberschrift varchar(250) NOT NULL,
   anreisser mediumtext NOT NULL,
   volltext text NOT NULL,
   PRIMARY KEY (nachricht_id)
);
```

7 Der Autor

Ulrich Hacke ist Jahrgang 1972 und arbeitet seit Mitte der achtziger Jahre mit Computern. Nach Abitur und Zivildienst absolvierte er das Studium der Religionspädagogik an der Ev. Fachhochschule Hannover, wo er sich in seiner Diplomarbeit intensiv mit den gesellschaftlichen Auswirkungen der Computertechnologie des Informationszeitalters beschäftigte.

Es folgte ein halbes Jahr stellvertretende Projektleitung für "kirche online" in Frankfurt am Main. 1997 begann er mit der Ausbildung zum Informatikassistenten (Softwaretechnologie) am b.i.b. Hannover. Anschliessend arbeitete er haupt- und freiberuflich im IT-Umfeld und gründete 1999 mit anderen das Unternehmen TRILOS IT-Dienstleistungen, dessen Geschäftsführer er heute ist.