Funktion und Parameter	Rückgabe	Beschreibung
`sqlite_open( string filename [, int mode [, string &error]])`	`resource`	Öffnet die Datenbank und erzeugt die Datei, wenn sie nicht existiert, mit dem Zugriffsmode *mode* (Unix-dateiparameter, beispielsweise 0666 (oktal)).
`sqlite_popen`	`resource`	Wie `sqlite_open`, aber persistent
`sqlite_query( resource handle, string query)`	`resource`	Fragt die Datenbank ab und gibt ein Handle auf die Ergebnisse zurück.
`sqlite_rewind( resource result)`	`bool`	Setzt den Zeiger auf die erste Reihe. Wird für die Schleifensteuerung verwendet.
`sqlite_seek( resource result, int row)`	`bool`	Setzt den Zeiger auf die Reihe *row*. Wird für die Schleifensteuerung verwendet.
`sqlite_udf_decode_binary (string data)`	`string`	Dekodiert Binärdaten, bevor diese an eine benutzerdefinierte Funktion gesendet werden.
`sqlite_udf_encode_binary (string data)`	`string`	Kodiert Binärdaten, bevor diese von einer benutzerdefinierte Funktion zurückgenommen werden.
`sqlite_unbuffered_query ( resource handle, string query)`	`string`	Wie `sqlite_query`, aber das Ergebnis wird nicht sofort komplett geholt, sondern erst, wenn es andere Funktion anfordern. Schneller bei einfachen sequenziellen Lesezugriffen.

Das folgende Beispiel zeigt, wie einige der Funktionen sinnvoll eingesetzt werden können.

11.6 SQLite ganz praktisch

Um den Umgang mit MySQL schnell zu erlernen, eignet sich ein kleines Datenbankprojekt. Vielleicht möchten Sie Ihren Besuchern regelmäßig ein paar Informationen zukommen lassen, die Sie im Web gefunden haben – ein Weblog oder kurz »Blog« entsteht.

11.6.1 Das Projekt vorbereiten

SQLite-Projekte sollten so geschrieben werden, dass Sie nicht auf einen Datenbankmanager angewiesen sind, jedenfalls solange, bis stabile Programme existieren. Deshalb wird das gezeigt Programm sowohl die Tabelle automatisch anlegen, als auch die nötigen SQL-Befehle zur Benutzung aufweisen.

Vorbereitung der Datenbank

Wenn das Skript benutzt wird, ist nur eine Vorbereitung notwendig: Das Verzeichnis, indem die Datenbankdatei abgelegt wird, ist vorher anzulegen und es müssen Schreibrechte existieren.

Zugriffssicherheit

Da im Blog sowohl Daten erfasst als auch ausgegeben werden, muss der Erfassungsteil geschützt werden. Dazu wird im Skript ein Kennwort kodiert. Damit niemand, der das Skript liest, das Kennwort im Klartext finden kann, nutzt man üblicherweise einen so genannten Hash. In diesem Fall das bekannte MD5 (Message Digest 5), dass auch für Unix-Kennwörter verwendet wird. PHP5 bietet dafür die Funktion `md5`.

Konfiguration

Als einziger Konfigurationsschritt soll der Name der Tabelle kodiert werden. Damit ist eine Anpassung an andere Umgebungen möglich. Die »Konfiguration« erfolgt ganz einfach durch eine Konstante.

11.6.2 Der Quellcode

Sie finden nachfolgend den Quellcode als ganzes – einschließlich HTML – und danach die Erklärung der wichtigsten Passagen.

Der Quellcode

Listing 11.6: Ein sehr einfaches Weblog (sqliteblog)

```
<html>
<head>
<title>SQLite Weblog</title>
<link rel="stylesheet" type="text/css" href="my.css">
</head>
<body>
<?php
define (PASSWORD, 'b8412671e5c89c647b691d6559129999');
define (TABLE, 'blog');

function makedate($date)
{
   return "Eintrag vom: $date";
}

$path = realpath('.') . '/blogger';
if ($db = sqlite_open("$path/blogger.db", 0666, $error))
{
   $tables = sqlite_array_query($db,
             "SELECT * FROM sqlite_master WHERE type='table'");
   $exists = false;
   foreach ($tables as $table)
   {
      if ($table['tbl_name'] == TABLE)
```

```
      {
         $exists = true;
         break;
      }
   }
   if (!$exists)
   {
       sqlite_query($db, 'CREATE TABLE ' . TABLE . '
                          (id INTEGER PRIMARY KEY,
                           blog TEXT,
                           source VARCHAR(100),
                           date DATETIME )');
   }
   if (isset($_POST['Send']))
   {
      $date = date('j.n.Y, h:i');
      sqlite_query($db, 'INSERT INTO ' . TABLE . '
                         (blog, source, date) ' .
                         "VALUES ('{$_POST['blog']}',
                         '{$_POST['source']}', '$date')");
   }
}
if (isset($_GET['admin']))
{
   if (md5($_GET['admin']) == PASSWORD)
   {
      echo <<<ADMINFORM
      <form action="{$_SERVER['PHP_SELF']}" method="post">
         Quelle:<br/>
         <input type="text" style="width:350px" name="source"/>
         <br/>
         Text: <br/>
         <textarea style="width:350px; height:100px" name="blog">
         </textarea>
         <br/>
         <br/>
         <input type="submit" name="Send" value="Eintragen"/>
      </form>
ADMINFORM;
   }
} else {
   sqlite_create_function($db, 'MAKEDATE', 'makedate', 1);
   $news = sqlite_array_query($db, 'SELECT blog, source,
           MAKEDATE(date) AS date FROM ' . TABLE . '
                          ORDER BY date DESC');
   foreach ($news as $blog)
   {
   echo <<<BLOG
   <div style="background-color:silver; font-weight:bold">
   Quelle: {$blog['source']} vom {$blog['date']}
   </div>
```

```
   <div style="background-color:white;color:blue;margin:10px">
   {$blog['blog']}
   </div>
BLOG;
   }
}
sqlite_close($db);
?>
</body>
</html>
```

Wie es funktioniert

Das Skript beginnt mit den beiden konfigurierbaren Parametern. Zuerst das Kennwort, dass den MD5-Hash des tatsaächlichen Kennworts darstellt. Die gezeigte Zeichenfolge entspricht dem Kennwort »Joerg«:

```
define (PASSWORD, 'b8412671e5c89c647b691d6559129999');
```

Die Eingabe des Kennworts erfolgt später als GET-Parameter (siehe dazu ➔ Abschnitt 9.2 *Daten per URL weiterreichen* ab Seite 213) im URL.

Der zweite konfigurierbare Wert ist der Name der Tabelle:

```
define (TABLE, 'blog');
```

Nun wird eine Funktion definiert, die später von SQLite heraus aufgerufen wird, um die Ausgabe des Datums zu modifizieren:

```
function makedate($date)
{
    return "Eintrag vom: $date";
}
```

Solche benutzerdefinierten SQL-Funktionen sind eines der raffiniertesten Konstrukte in PHP5 und SQLite, denn sie »verweben« Datenbank und Code und erlauben kompakte Lösungen.

Der Pfad zur Datenbank muss immer ein absoluter Pfad sein. Deshalb wird die Funktion `realpath` verwendet, um aus dem aktuellen Verzeichnis den Pfad zu ermitteln:

```
$path = realpath('.') . '/blogger';
```

Dann wird die Datenbank geöffnet und ein Handle (in *$db*) erzeugt. Existiert die Datenbank nicht, wird sie erzeugt:

```
if ($db = sqlite_open("$path/blogger.db", 0666, $error))
```

Nun ist zu prüfen, ob die Tabelle bereits existiert. Dazu wird die Master-Tabelle abgefragt:

```
$tables = sqlite_array_query($db,
          "SELECT * FROM sqlite_master WHERE type='table'");
```

Um den Tabellennamen zu erhalten, wird das Array der Tabelleninformationen durchlaufen und bei Erfolg die Variable *$exists* auf `TRUE` gesetzt:

```
$exists = false;
foreach ($tables as $table)
{
   if ($table['tbl_name'] == TABLE)
   {
      $exists = true;
      break;
   }
}
```

Falls die Tabelle noch nicht existiert, wird sie angelegt:

```
   if (!$exists)
   {
       sqlite_query($db, 'CREATE TABLE ' . TABLE . '
                             (id INTEGER PRIMARY KEY,
                              blog TEXT,
                              source VARCHAR(100),
                              date DATETIME )');
   }
```

Alle weiteren Operationen gehen davon aus, dass die Tabelle existiert.

Nun erfolgt die Auswertung des Administrationsformulars, das weiter unten noch erzeugt wird. Dazu wird geprüft, ob die Sendeschaltfläche gedrückt wurde. Mehr Informationen zu Formularen finden Sie im ➔ Abschnitt 9.1.2 *Formularelemente analysieren* ab Seite 202.

```
   if (isset($_POST['Send']))
   {
```

Wurde das Formular ausgefüllt, wird das aktuelle Datum ermittelt. Mehr Informationen zu Datumsoperationen finden Sie im ➔ Abschnitt 13.4.2 *Datumsfunktionen* ab Seite 337:

```
      $date = date('j.n.Y, h:i');
```

Dann werden die Daten in die Datenbanktabelle eingetragen:

```
      sqlite_query($db, 'INSERT INTO ' . TABLE . '
                         (blog, source, date) ' .
                          "VALUES ('{$_POST['blog']}',
                          '{$_POST['source']}', '$date')");
   }
}
```

Der Rest des Skripts beschäftigt sich mit dem Eingabeformular und der Ausgabe der Daten. Das Eingabeformular wird erzeugt, wenn an den URL die Parameter *admin=Kennwort* angehängt wurden, wobei statt »Kennwort« das in der MD5-Form kodierte Wort zu verwenden ist.

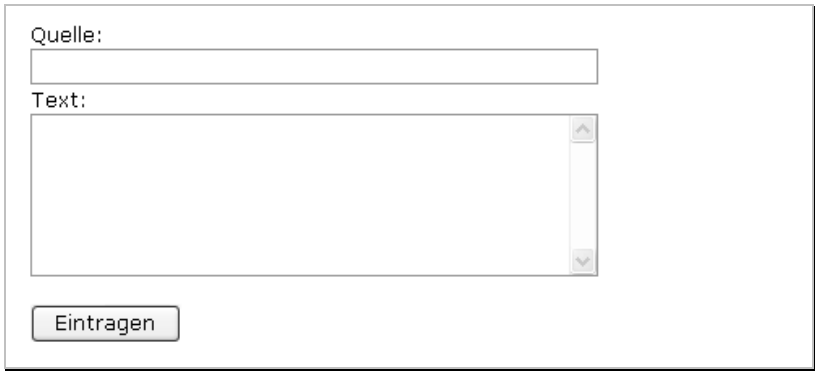

Abbildung 11.6: Administration des Weblogs

Zuerst wird jedoch geprüft, ob überhaupt der Parameter *admin* existiert:

```
if (isset($_GET['admin']))
```

Dann erfolgt der Vergleich mit dem gespeicherten Hash:

```
   if (md5($_GET['admin']) == PASSWORD)
```

Stimmt alles, wird die Administrationsform erzeugt:

```
   echo <<<ADMINFORM
   <form action="{$_SERVER['PHP_SELF']}" method="post">
      Quelle:<br/>
      <input type="text" style="width:350px" name="source"/><br/>
      Text: <br/>
      <textarea style="width:350px; height:100px" name="blog">
      </textarea>
      <br/>
      <br/>
      <input type="submit" name="Send" value="Eintragen"/>
   </form>
ADMINFORM;
```

Ist kein Kennwort angegeben worden, wird die Ausgabe erzeugt:

```
} else {
```

Dazu wird zuerst die benutzerdefinierte Funktion in SQLite bekannt gemacht. Definiert wurde `makedate` oben bereits:

```
   sqlite_create_function($db, 'MAKEDATE', 'makedate', 1);
```

Jetzt erfolgt die Abfrage, wobei im SQL-Befehl die Funktion `MAKEDATE` benutzt werden kann:

```
   $news = sqlite_array_query($db,
           'SELECT blog, source, MAKEDATE(date) AS date
            FROM ' . TABLE . '
            ORDER BY date DESC');
```

Das Ergebnis ist ein Array, dass mit `foreach` ausgegeben werden kann:

```
   foreach ($news as $blog)
   {
   echo <<<BLOG
```

```
   <div style="background-color:silver; font-weight:bold">
   Quelle: {$blog['source']} vom {$blog['date']}
   </div>
   <div style="background-color:white; color:blue; margin:10px">
   {$blog['blog']}
   </div>
BLOG;
   }
}
```

Am Ende ist die Datenbank noch zu schließen:

```
sqlite_close($db);
```

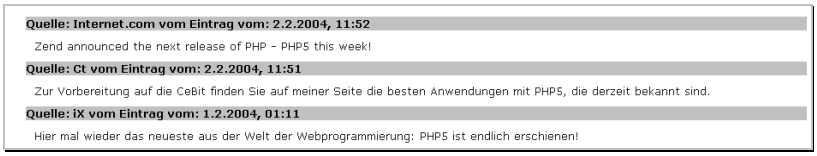

Abbildung 11.7: Ein einfaches Weblog

11.6.3 Diskussion

SQLite ist für viele Fälle nicht nur sehr einfach einzusetzen, sondern auch clever programmierbar. Allein die benutzerdefinierten Funktionen erlauben sehr kompakte Lösungen – ein Feature, dass MySQL schmerzlich vermissen lässt. Es lohnt sich also unbedingt, sich mit SQLite auseinanderzusetzen, wenn ein Projekt ausschließlich für PHP5 geschrieben wird. Auf eine Abwärtskompatibilität zu PHP4 muss man hier freilich verzichten.

Das vorgestellte Skript ist sehr primitiv. Sie sollten es vor dem ersten Einsatz noch um Fehlerroutinen erweitern, die mögliche Fehlzustände abfangen. Außerdem sollte vermieden werden, dass Einträge doppelt eingetragen werden. Und nicht zuletzt sollte eine Chance bestehen, die Einträge wieder zu entfernen oder dies automatisch nach Ablauf einer gewissen Zeit erledigen zu lassen.