PHP und Centura SQLBase

Laut einer aktuellen Information in der Gupta Newsgroup[1] arbeiten mehr als eine Million Anwender weltweit mit Gupta bzw. Centura SQLBase. Sicherlich wird die SQLBase bisher hauptsächlich zusammen mit dem Gupta Team Developer eingesetzt. Aber kann die »Base« auch mit PHP?

von Thomas Wiedmann

Gesetzt den Fall, Sie wünschen sich ein Intranet. Und glauben Sie mir, das ist ein gar nicht so ungewöhnlicher Gedanke. Zur Zeit laufen ihre Gupta Anwendungen - einschließlich der SQLBase - in einem Windowsnetzwerk. Ein Intranet wiederum ist browserbasiert - folglich ziemlich betriebssystemneutral - und für browserbasierte Anwendungen gilt, häufig wird PHP eingesetzt. Dies mündet in der zentralen Frage**: »Wie komme ich mit PHP an meine SQLBase dran?«**. Dass es geht, zeige ich Ihnen gleich hier (Bild 1). Wie es genau geht, ist das Thema des folgendes Artikels.

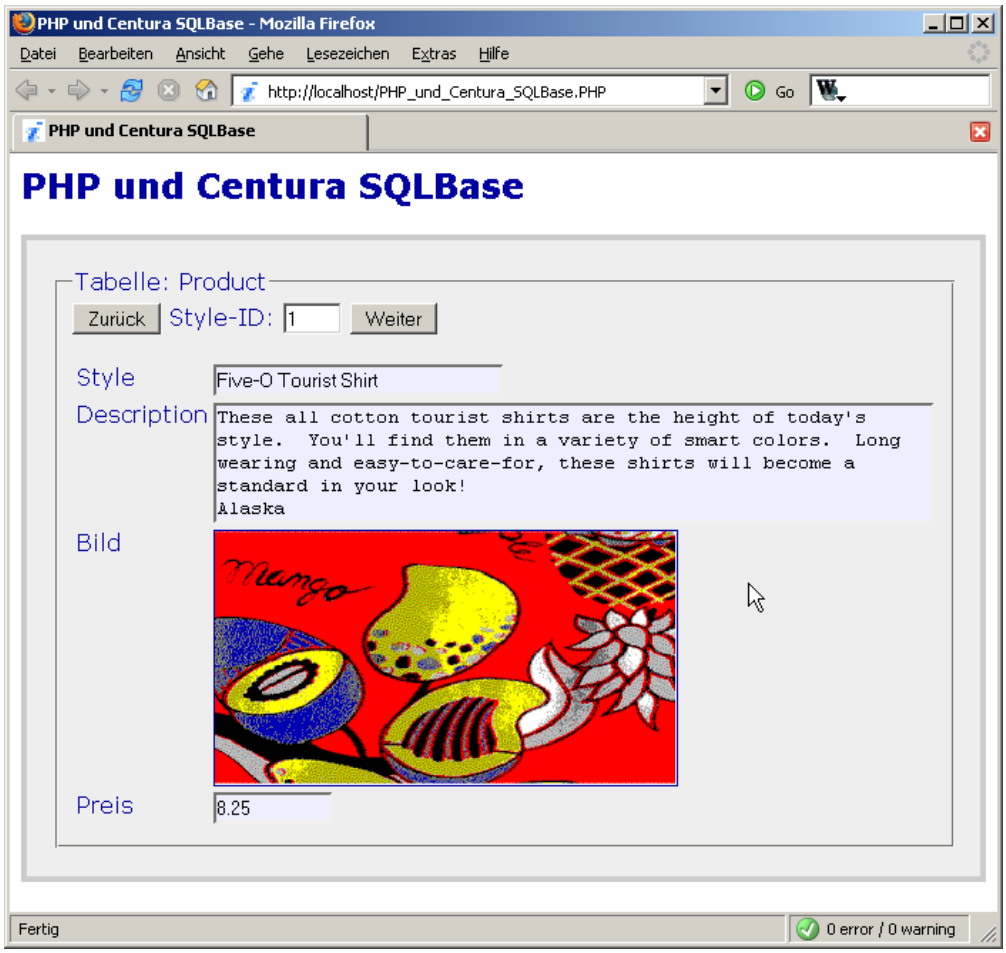

Bild 1: Die Tabelle PRODUCT der Island-Datenbank wird angezeigt.

Als SQLBase Nutzer kennen Sie bestimmt die Beispieldatenbank ISLAND und davon die Tabelle PRODUCT. Sie sehen, es ist für PHP kein Problem, die vorhandenen Daten auszulesen und mit Hilfe von HTML (**H**yper**T**ext **M**arkup **L**anguage) plus ein bißchen CSS (**C**as**c**ading **S**tylesheet) übersichtlich darzustellen. Die Verbindung zur Datenbank wird dabei über ODBC (**O**pen **D**ata**b**ase **C**onnectivity) aufgebaut. Das Projekt wird erst einmal mit Windows aufgesetzt. Was Sie dazu brauchen sind: ein lauffähiger Apache Webserver und

dazu eine aktuelle und lauffähige PHP Version. Beispielsweise PHP v 5.0.4. Der Download der neuesten Version hier möglich [2].

Übersicht und Installation

Folgende Module sind im Einsatz:

- Windows 2000 Professional
- Centura SQLBase 7.5.1 (Developer Edition)
- Apache 1.3.28
- PHP 5.0.4

Für die Entwicklung habe ich erst einmal alles auf einem PC installiert. Den Apache plus PHP setze ich als lauffähig voraus. Für die Verbindungsaufnahme von PHP zur SQLBase wird ODBC eingesetzt. Damit SQLBase ODBC akzeptiert, muß die berühmte SQL.INI angepaßt und ergänzt werden. Welche SQL.INI, lokal oder Server? Nur auf dem SQLBase Server und gegebenenfalls dort wo PHP läuft, falls Sie einen separaten WebServer für Ihr Intranet planen. PHP benötigt in diesem Falle eine SQLBase Clientinstallation.

```
SQL.INI (Ausschnitt + Ergänzungen):
[...]
[win32client]
clientname=Win32Client

[win32client.dll]
comdll=sqlodb32

[odbcrtr]
remotedbname=ISLAND,DSN=ODBC_ISLAND
[...]
```

Mit comdll=sqlodb32 wird der ODBC-Router aktiviert. Im Abschnitt [odbcrtr] erfolgt die Zuordnung des sogenannten DSN ODBC_ISLAND zur eigentlichen SQLBase Datenbank ISLAND. Um die Änderungen in der SQL.INI zu aktivieren, muß der SQLBase Server neu gestartet werden.

ODBC Data Source Administrator

Nach den Ergänzungen in der SQL.INI ist als nächstes ein neuer DSN (**D**ata **S**ource **N**ame) notwendig. Dazu öffnen wir den Dialog »Datenquellen (ODBC)« unter Systemsteuerung / Verwaltung und klicken unter System-DSN den Knopf [Add]. Ist der SQLBase Client mit ODBC Support installiert, finden Sie einen Eintrag zur SQLBase, diesen markieren Sie. Beispielsweise so (Bild 2) und wählen [Fertig stellen]. Anschließend öffnet sich der eigentliche Setup-Dialog mit dem Namen »ODBC SQLBase Driver Setup«. Hinter dem komplizierten Titel verbirgt sich eine relativ harmlose Maske (Bild 3). Darin tragen Sie die neue DSN ein.

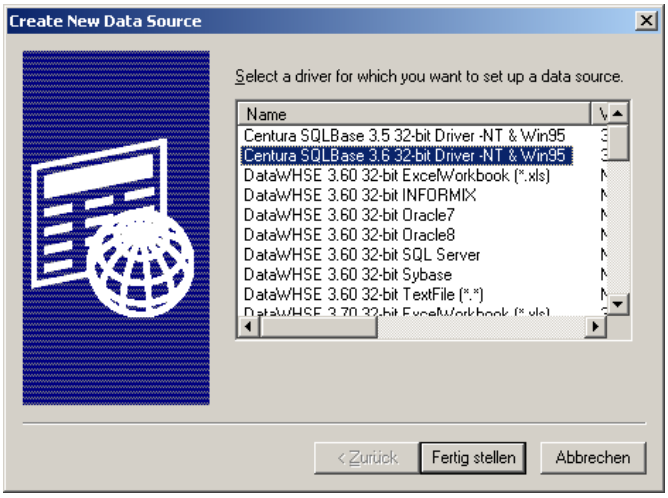

Bild 2: Auswahl des verfügbaren ODBC-Treibers (Centura SQLBase 3.6)

Als Data Source Name schlage ich mal ODBC_ISLAND vor. Der Name ist recht sprechend und zeigt deutlich, dass es sich hier um eine ODBC Verbindung auf die Datenbank ISLAND handelt. Als ersten Test empfiehlt sich [Test Connect] auszuprobieren. Sehen Sie nun einen weiteren Dialog mit dem Namen »Logon to SQLBase«, hat dieser Installationsschritt schon mal funktioniert.

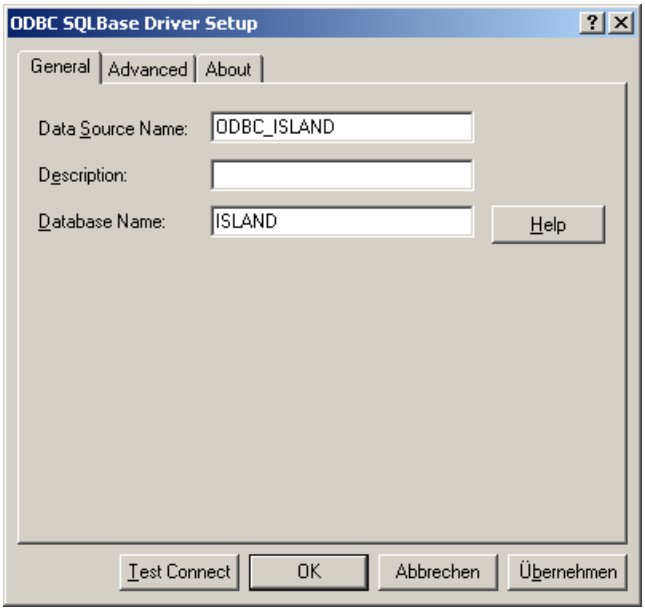

Bild 3: ODBC SQLBase Driver Setup - neue DSN anlegen

Als dritter und letzter Schritt fehlen jetzt noch Einträge auf der Karteikarte »Advanced« (siehe Bild 3). Dort wird der Servername hinterlegt, auf der die SQLBase läuft. In meiner Testinstallation lautet der Name wie üblich auf »Server1« (Bild 4). Ist auch dieser Eintrag erfolgt und mit [Ok] bestätigt, steht dem ersten Test nichts mehr im Wege. Klicken Sie auf [Test Connect] und ergänzen im Dialog »Logon to SQLBase«, Username und Passwort. Nach [Ok] erscheint hoffentlich ein freundliches »Connection established«.

Falls nicht, wenden Sie sich am besten an den Administrator Ihres Vertrauens.

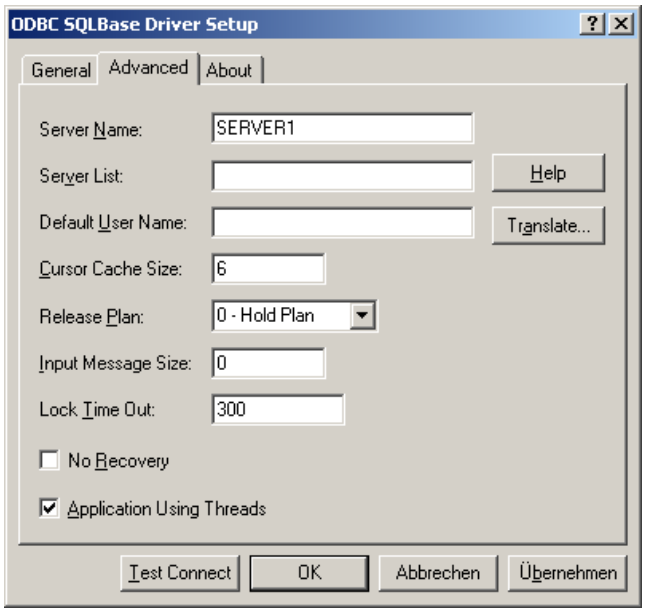

Bild 4: Advanced Dialog beim ODBC Setup

PHP.INI anpassen ?

Ja, die PHP.INI gibt es auch und ist meistens im WinNT-Verzeichnis des Server angesiedelt. Und anpassen? Nein, anpassen muß man sie nicht, der sogenannte ODBC Support ist bei der Windows-Version von PHP bereits per Default aktiviert (Build in). So, jetzt ist es an der Zeit unser erstes Beispiel auszuprobieren. Standesgemäß ein »HelloWorld« von PHP an die SQLBase.

Hello World!

Jetzt ist es an der Zeit, unser kleines »HelloWorld.php« Skript in das *DocumentRoot*-Verzeichnis des Apache Servers zu kopieren, einen Browser zu starten und das Skript aufzurufen. Im Falle einer lokalen Installation lautet die URL dazu: *http:\localhost\HelloWorld.php*. Im folgenden Listing (1) sehen Sie den Verbindungsaufbau zur Datenbank.

```
Listing 1: (HelloWorld.php)

[...]
/**
 * Verbindungsaufbau mit dem gültigen Kennwort
 */
 $hSqlA = odbc_connect('ODBC_ISLAND','SYSADM','SYSADM');
 if ($hSqlA == false) {
  echo '<br>ABBRUCH odbc_connect failed.. <br>\n';
  echo odbc_errormsg();
  die();
 }
[...]
```

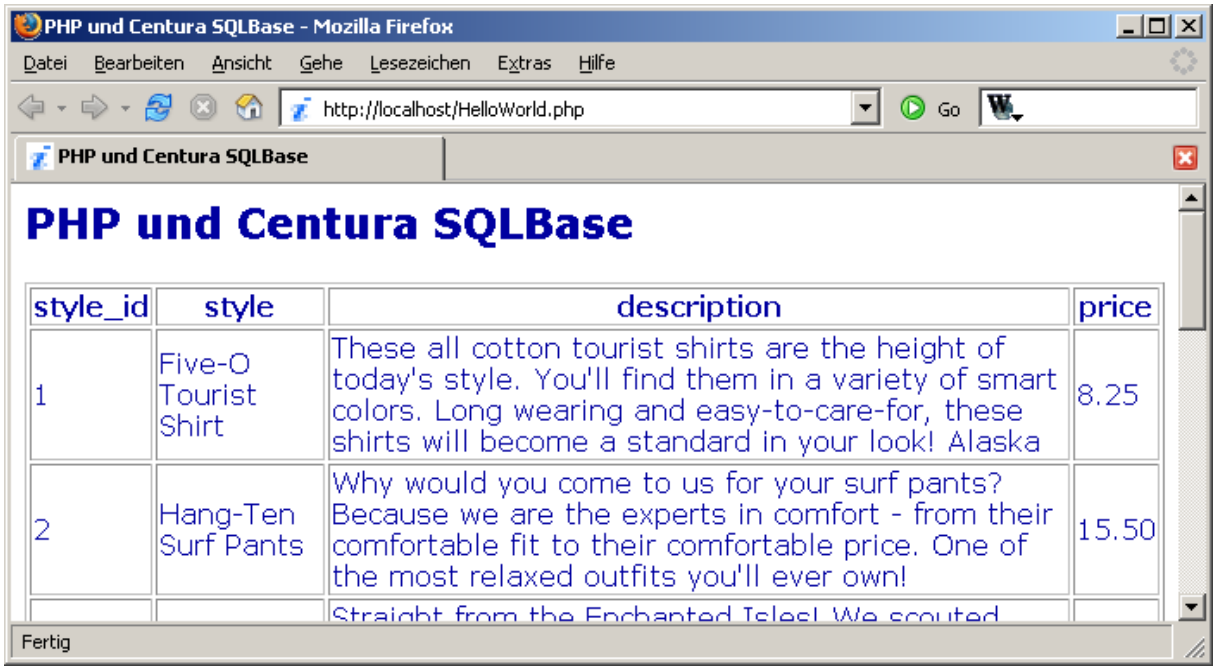

Bild 5: HelloWorld.php zeigt den Inhalt der SQLBase Tabelle »PRODUCT«

Na bitte, bei mir funktioniert es und hoffentlich bei Ihnen auch. »HelloWorld.php« läuft und liefert in Tabellenform den Inhalt der Tabelle PRODUCT. Nun ist es an der Zeit hinter die Kulissen zu schauen. Der notwendige SQL sieht so aus:

```
Listing 2: (HelloWorld.php)

[...]
/**
 * SELECT aufbauen und ausführen
 */
 $sQuery = '';
 $sQuery = $sQuery.'SELECT style_id, style, description, price ';
 $sQuery = $sQuery.'  FROM sysadm.product ';

 $hCursor = odbc_exec($hSqlA,$sQuery);
  if ($hCursor) {
[...]
```

Soweit nichts neues. Was Sie vielleicht vermissen ist das INTO Kommando, aber dazu später mehr. Odbc_exec() führt den SQL-Befehl aus. Ist dieser korrekt, wird ein Cursor *$hCursor* zurück gegeben. Damit haben wir nun auch Zugriff auf die Ergebnismenge, den Resultset (Ergebnismenge).

```
Listing 3: (HelloWorld.php)

[...]
  /**
  * Anzahl der Ergebnisspalten ermitteln
  */
  $nCols = odbc_num_fields($hCursor);

  /**
  * Tabellen-Header erzeugen
  */
  $sTable = $sTable.'<tr>';
  for ($nCol = 1; $nCol <= $nCols; $nCol++) {
   $sTable = $sTable.'<th>'.odbc_field_name($hCursor,$nCol).'</th>';
  }
  $sTable = $sTable.'</tr>';
[...]
```

Im nächsten Schritt wird der Tabellenkopf erzeugt. Die Funktion odbc_num_fields() liefert die Anzahl der Ergebnisspalten. Nun kann per Schleife und odbc_field_name() jeder Spaltenname der Ergebnismenge ermittelt und der HTML Tabellenkopf mit Hilfe der Tags <th> und </th> aufgebaut werden. Weitere Informationen zu HTML liefert die sehr gute und kostenlose SELFHTML Referenz von Stefan Münz [3].

```
Listing 5: (HelloWorld.php)

[...]
  /**
  * Tabellendaten einlesen
  */

  While ($Row = odbc_fetch_row($hCursor)) {

   $sTable = $sTable.'<tr>';
   for ($nCol = 1; $nCol <= $nCols; $nCol++) {
    $sTable = $sTable.'<td>'.odbc_result($hCursor,$nCol).'</td>';
   }
   $sTable = $sTable.'</tr>';

  }
[...]
```

Im Listing 5 werden schlussendlich die Spaltenwerte abgeholt und in die <td></td> Tabellenzellen eingesetzt. Für Gupta-Programmierer ist hier noch vielleicht wesentlich: Der String-Concat ist der ».« (Punkt) und PHP macht keinen echten Unterschied bei den Datentypen. Die eigentliche Ausgabe der Tabelle *<?php echo $sTable; ?>* erfolgt etwas später im Skript. Damit ist der Programmcode wieder von GUI getrennt (Listing 6). Ein wichtiges Kriterium, um nicht gleich mit »Spagetticode« zu beginnen.

```
Listing 6: (HelloWorld.php)

[...]
<body>
 <h2>PHP und Centura SQLBase</h2>
 <?php echo $sTable; ?>
</body>
[...]
```

Das HelloWorld – Beispiel hat gezeigt, wie einfach es letzlich ist, Datenbankinhalte in Tabellenform in den Browser zu bringen. Allerdings ißt das Auge des Benutzer mit, das bedeutet, die Darstellung muß mit etwas CSS aufgemöbelt werden, damit der verwöhnte Windowsbenutzer sie akzeptiert. Zudem fehlt noch die Interaktion. Der Anwender möchte bestimmte Daten abfragen, dazu müssen Eingaben – ala Google – möglich sein.

Daten und Bilder

Im nächsten Schritt geht es darum, die in der Tabelle PRODUCT hinterlegten Bilder auszugeben. Ein Blick auf die Tabellenstruktur (Bild 6) zeigt, es gibt da eine Spalte PICTURE mit dem Datentyp LONGVAR. Ein Versuch den Inhalt mit SQLTalk anzuzeigen, scheitert logischerweise kläglich, zeigt aber auch, der Inhalt ist binär. Übrigens, diese kleine Tabellen-dokumentation kann mit dem Pro-gramm SQLBaseHtmlDoc.zip erzeugt werden. Das GTD-Programm steht inclusive Source kostenlos zum Download auf meiner Website zur Verfügung[4].

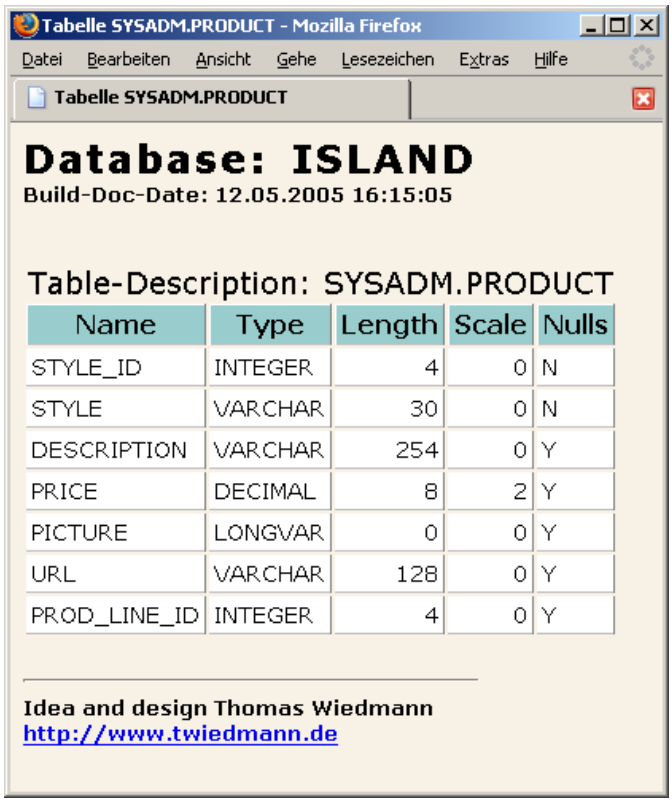

Bild 6: Tabellenstruktur SYSADM.PRODUCT

Beginnen wir damit die »normalen« Spalten Style, Description und Price auszugeben. Mit den bisherigen Informationen sollte dies kein Problem für Sie sein. Aber diesmal ist etwas mehr Komfort und Interaktion für den Anwender gewünscht. Das Ergebnis der Bemühungen war bereits eingangs in Bild 1 zu sehen. Der Anwender kann eine bestimmte Style_Id eingeben und jeweils einen Satz vor oder zurück blättern.

```
Listing 7: (PHP_und_Centura_SQLBase.PHP)
[...]
/**
 * Formular auswerten
 */
 if (isset($_POST['StyleId'])) {
  $nStyleId = (INT) $_POST['StyleId'];
 }

 if (isset($_POST['next'])) {
  $nStyleId = $nStyleId +1;
 }

 if (isset($_POST['prev'])) {
  $nStyleId = $nStyleId -1;
  if ($nStyleId < 1) {
   $nStyleId = 1;
  }
 }
[...]
```

Listing 7 zeigt das wesentlich Neue. Die Auswertung der Eingabefelder nach einem [Submit] durch den Anwender. Submit bedeutet, der Anwender hat auf [Ok] oder einen der Knöpfe

[Zurück] oder [Weiter] geklickt. Das Formular wird mit der »POST« Methode vom Browser zurück an den Server geschickt und dort von dem PHP-Programm (*PHP_und_Centura_SQLBase.PHP*) – quasi von sich selbst – wieder ausgewertet. Die – zugegeben sehr einfach gehaltene - Auswertung erfolgt in drei Schritten.

- Existiert die Variable $_POST['StyleId'] wird der Inhalt ausgelesen.
- Existiert die Variable $_POST['next'], hat der Anwender [Zurück] geklickt
- Existiert die Variable $_POST['prev'], hat der Anwender [Weiter] geklickt

Alles weitere ergibt sich aus dem PHP-Source und ist nichts neues mehr für Sie. Neu eingeführt sind allerdings die Input HTML Befehle, die erst eine Interaktion zulassen. Auch die [Knöpfe] sind nur eine spezielle Art eines Input Elementes.

```
Listing 8: (PHP_und_Centura_SQLBase.PHP)
[...]
<form method="post"
      action="<?php echo $_SERVER['PHP_SELF'] ?>"
      accept-charset="ISO-8859-1">
  <legend>Tabelle: Product</legend>
  <input type="submit" name="prev" value="Zur&uuml;ck" />
  Style-ID:
  <input type="text" name="StyleId" size="3" maxlength="2"
    class="edit" value="<?php echo $nStyleId ?>" />
  <input type="submit" name="next" value="Weiter" />
[...]
```

Mit Listing 8 sind wir schon ziemlich tief in die HTML-Technik eingedrungen. Ausführlich über HTML und CSS können Sie sich bei Stefan Münz [2] informieren. Jedes hier gezeigte *<input..>* - Element enthält innerhalb der spitzen Klammer spezielle Attribute, mit denen - unter anderem - der eigene Name, der eigene Inhalt (*value="<?php echo $nStyleId ?>"*) und maximale Eingabelänge festgelegt werden kann. Interessant ist das Attribut *class*. Damit wird auf eine CSS-Klasse verwiesen, mittels der die Darstellung (GUI) angepasst werden kann. CSS zu nutzen ist existentiell. Nur so ist es möglich, einfach und schnell die Oberflächendarstellung den Wünschen des Anwenders anzupassen. Listing 9 zeigt die notwendigen CSS Befehle, um Bild 1 zu gestalten.

```
Listing 9: (PHP_und_Centura_SQLBase.PHP)
[...]
<style type="text/css">
  body { background-color:#FFFFFF; font-family: verdana,arial,helvetica;
         color: #0000A0; }
  form { padding:15px; border:4px solid #CCCCCC; background-color:#EEEEEE; }
  legend { font-size: 1em;  }
  fieldset{ margin: 5px 5px 5px 5px;  }
  input.edit { color:#000000; background-color:#FFFFFF; }
  input.disable { color:#000000; background-color:#EEEEFF;  }
  textarea.disable { color:#000000; background-color:#EEEEFF;  }
  td { vertical-align:top; }
</style>
[...]
```

Um einem HTML-Formularfeld einen Vorgabewert zu geben, muß das Attribut »value« gefüllt werden. Da die Feldinhalte nicht statisch, sondern dynamisch sind, muß die Ausgabe quasi zur Laufzeit durch PHP erfolgen. Der »echo« Befehl von PHP ist dafür zuständig. (siehe Listing 10).

```
Listing 10: (PHP_und_Centura_SQLBase.PHP)
[...]
<input type="text"
       name="Style"
       size="30"
       class="disable"
       value="<?php echo $sStyle; ?>"
       readonly="readonly" />
```

Das Bild aus der Datenbank

Bleibt noch das Problem mit dem Produktbild. Typischerweise werden in HTML statische Bilder angezeigt, also Bilder, die als JPG- oder GIF-Datei innerhalb des Filesystem vorliegen. Da das Bild aber innerhalb der Datenbank als Inhalt einer Tabellenspalte vorliegt, kommen die üblichen Vorgehensweisen nicht in betracht. Zwei Lösungsmöglichkeiten stehen somit zur Diskussion:

1. Das Bild per PHP aus der Datenbank lesen und in ein temporäres File speichern. Wäre möglich, aber wann wird das Bild wieder gelöscht? Denken Sie dabei auch an die Möglichkeit, dass der Anwender im Browser vor- und zurück blättern kann.
2. Das Bild wird mit PHP aus der Datenbank gelesen und in ein Bilddokument umgewandelt. Damit entsteht das Bild innerhalb dem Browsercache. Das Blättern im Browser funktioniert problemlos und gelöscht wird es irgendwann automatisch.

Aus diesen Überlegungen heraus, ist der zweite Ansatz vermutlich der bessere, allerdings ist es trotzdem etwas verzwickt.

```
Listing 11: (PHP_und_Centura_SQLBase2.PHP)
[...]
/**
 * Ergebnis einlesen
 */
 IF ($Row = odbc_fetch_row($hCursor)) {
  $sPicture = odbc_result($hCursor,3);
 }
[...]
```

Das Bild kann problemlos aus der Datenbank gelesen werden. Der binäre Inhalt landet in der String-Variablen *$sPicture.* Jetzt könnte man auf die Idee kommen, den Inhalt von *$sPicture* einfach per *<?php echo $sPicture; ?>* oder *<?php echo '' ?>* auszugeben. Das Ergebnis wird Sie und die Anwender allerdings etwas verwirren. Das liegt zum einen daran, das der HTML-Befehl *src="Bild.gif"* eine Grafikdatei mit oder ohne Pfadangabe erwartet. Zum anderen ergänzt der GTD offensichtlich beim Speichern von Bilder den Header um eigene Informationen. Dies können Sie mit folgendem SQL-Befehl innerhalb SQLTalk überprüfen.

```
Listing 12:

SELECT picture FROM product WHERE style_id = 1;

PICTURE
=======
Centura:GIF
CF|82dÈ,•?f´ˆQ%1–•/fÜ8“ãË‹1[Òt¹“¦I”••••9±âPŠÆ†
[...]
```

Daher muß von dem eingelesen Binärstring ein Teil entfernt und der Rest anschließend in ein Image umgewandelt werden. Für PHP stellt das kein Problem dar.

```
Listing 13: (PHP_und_Centura_SQLBase2.PHP)
[...]
/**
 * Centura:GIF entfernen bis "GIF89"
 */

 if ($sPicture != '') {

  $sPicture = strstr($sPicture,'GIF89');

  $im = imagecreatefromstring($sPicture);
  if ($im !== false) {
    header('Content-Type: image/gif');
```

```
    imagegif($im);

  }
 }
[...]
```

Zuerst werden alle Zeichen vor der Zeichenkette 'GIF89' entfernt und der Datenstring anschließend mittels *imagecreatefromstring()* umgewandelt und mit *imagegif()* an den Browser geschickt. Der Befehl *header()* teilt dem Browser mit, der Inhalt des Dokumentes ist ein GIF-Bild. Damit ist ein neues Dokument mit dem GIF-Bild als Inhalt entstanden. Das ist der erste Schritt zum Ziel. Wenn Sie testweise im Browser folgenden URL ausführen, sollten Sie das Bild 7 sehen. Bitte achten Sie auf die Parameterübergabe **id=1,** die die angeforderte *Style_Id* darstellt..

URL: http://localhost/PHP_und_Centura_SQLBase2.PHP?id=1

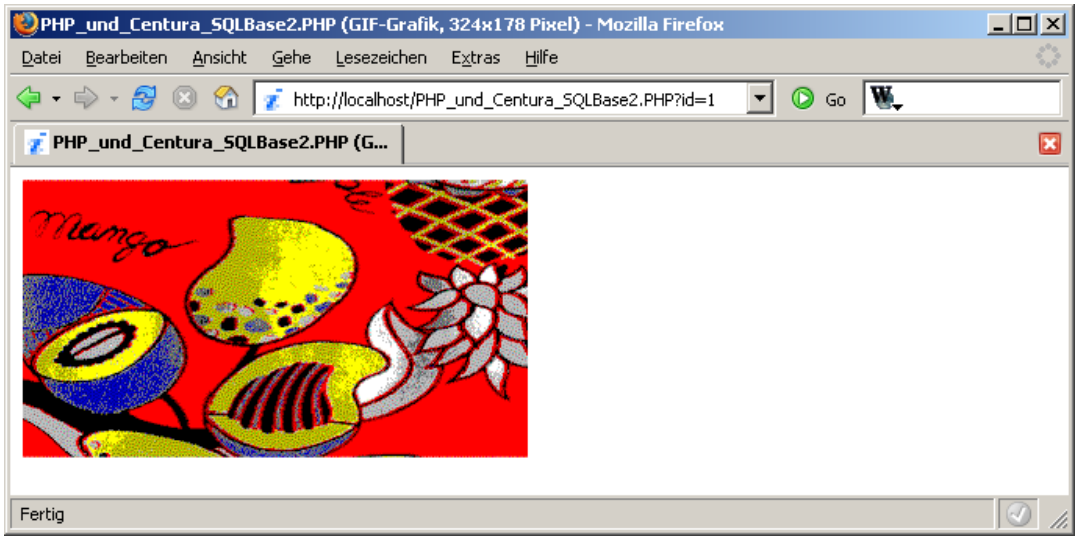

Bild 7: Ein Bild aus der Datenbank darstellen (PHP_und_Centura_SQLBase2.PHP)

Sollten Sie hierbei Fehlermeldungen wie:

> **Fatal error**: Call to undefined function imagecreatefromstring() ..

bekommen, müssen Sie in der PHP.INI die Grafikbibliothek GD2 *extension=php_gd2.dll* eintragen *bzw.* aktivieren. Änderungen in der PHP.INI werden erst nach einem *restart* des Apache Webserver aktiviert. Am besten schließen und starten Sie den Browser ebenfalls.

Nachdem das in der Datenbank gespeicherte Bild wieder angezeigt werden kann, muß nun noch das Bild innerhalb des Formular ausgegeben werden. Das ist zum Glück einfach, wenn auch raffiniert. Das PHP-Skript *PHP_und_Centura_SQLBase2.PHP* erzeugt ein Dokument, bestehend aus einem Bild. Damit das Bilddokument in das bestehende Formular von Skript *PHP_und_Centura_SQLBase.PHP* kommt, muß es einfach anstatt eines »echten« Bildes eingebunden werden:

```
Listing 14: (PHP_und_Centura_SQLBase.PHP)
[...]
    <tr>
     <td>Bild
     </td>
     <td><?php if ($nStyleId > 0)
           echo '<img border="1"
           src="PHP_und_Centura_SQLBase2.PHP?id='.$nStyleId.'"
           alt="Bild" />'; ?>
```

```
    </td>
   </tr>
[...]
```

Dem *src* – Attribut wird anstatt eines echten Bildes, die Ausgabe des PHP-Sktiptes zugewiesen und was wunder, es tut. Eigentlich sind wir hiermit am Ziel. Die PHP-Skripte liefern nur das Ergebnis analog Bild 1. Aber um richtig und professionell mit PHP und SQLBase arbeiten zu können, bedarf es noch paar weiterer Tipps.

PHP – Datenbankzugriffsklasse

Bereits ab der Version 4 kann PHP recht gut mit Objekten und Klassen umgehen. Wenn Sie sich jetzt nochmal das Schichtenmodell der Softwareentwicklung vor Augen führen, werden Sie feststellen – es macht Sinn, die Datenbankzugriffe in eine eigene Klasse zu kapseln und das HTML-Layout (Präsentationsschicht) gleichzeitig von der Datenverwaltung zu trennen [4]. Damit das Thema nicht gar zu trocken wird, habe ich Ihnen ein komplexes Beispiel erstellt. Damit können Datensätze gelesen, geändert und gelöscht werden. Ein bißchen Feldvalidierung ist auch vorgesehen. Also all das, was der harte Alltag in der Praxis so – natürlich komplexer - vorsieht.

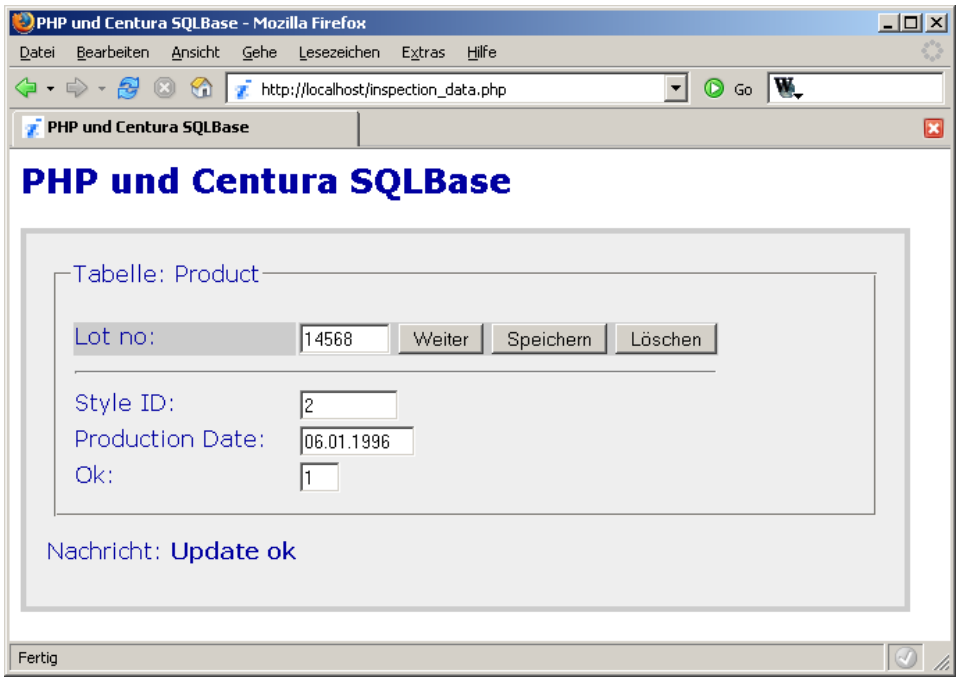

Bild 8: Die Maske (Präsentationsschicht) für die Tabelle INSPECTION_DATA (inspaction_data.php)

Die Tabelle *INSPECTION_DAT*A stammt ebenfalls aus der *ISLAND* Datenbank und damit liegen Ihnen diese Daten vor. Die Maske ist selbsterklärend. Mit [Weiter] wird zum nächsten Datensatz geblättert. Grundlage hierfür ist der Inhalt des Schlüsselfeldes *Lot No*. [Weiter] erhöht den aktuellen Inhalt von *Lot No* und versucht anschließend den Datensatz zu lesen. So weit, so trivial, denn sollte eine Lücke in der Zahlenreihe sein, funktioniert der Trick nicht mehr. [Speichern] führt einen UPDATE mit den Maskenfeldern aus, während [Löschen] den aktuellen Datensatz – ohne Rückfrage – löscht.

```
Listing 15: (inspection_data.php)
[...]
/**
 * Klasse "inspection_data" einbinden.
 * Das File muß existieren, sonst gibt es einen Abbruch.
 */
 require_once('inspection_data_class.php');

 /**
 * Object aus der Klasse "inspection_data" erzeugen
 */
 $oInspection_data = new inspection_data();
[...]
```

Aber eigentlich wollte ich Ihnen die Datenzugriffsklasse vorstellen. In Listing 15 wird ein neues Objekt mit *new inspection_data()* erzeugt. Die Variable *$oInspection_data* hat damit alle Fähigkeiten geerbt, um die Tabelle INSPECTION_DATA zu verwalten. Dazu gehört:

- Set / Get
- Validate
- Select
- Update
- Delete

Im Falle von [Speichern] sieht das wie folgt aus (inspection_data.php):

- Die Inhalte der Input-Felder werde nach dem POST in eigene Variablen ausgelesen
- Mit *Set()* z.B. *$oInspection_data->SetsLot_no($sLot_no)* werden die Inhalte an die Klasse übergeben.
- Nun wird *if ($oInspection_data->Validate() == false)* aufgerufen und ausgewertet.
- Geht alles glatt, erfolgt der eigentlich Update *if ($oInspection_data->Update($hSqlA))* und abschließend der Commit *$oInspection_data->Commit($hSqlA) .*

Das PHP Skript *inspection_data.php* steuert den Datenbankzugriff quasi aus der »Ferne«. Der Tabellenzugriff ist gekapselt. Die Vorteile dieser oder ähnlicher Techniken sind schon häufig beschrieben worden [4].

```
Listing 16: (inspection_data_class.php)
[...]
  /**
  * UPDATE
  */
  function Update ($p_hConnection = 0) {

   $sUpdate1 = '';
   $sUpdate1 = $sUpdate1.'UPDATE sysadm.inspection_data SET ';
   $sUpdate1 = $sUpdate1.' style_id = '.$this->nStyle_id.', ';
   $sUpdate1 = $sUpdate1.' production_date = \''.$this->dtProduction_date.'\', ';
   $sUpdate1 = $sUpdate1.' ok = \''.$this->sOk.'\' ';
   $sUpdate1 = $sUpdate1.' WHERE lot_no = \''.$this->sLot_no.'\'';

   /**
   * ODBC_EXEC ausführen und im Fehlerfall ein ROLLBACK auslösen
   */
   $ret = odbc_exec($p_hConnection,$sUpdate1);
   if ($ret == false) {
    odbc_exec($p_hConnection,'ROLLBACK');
    return false;
   } else {
    /* update ok */
    return true;
   }
  } /* Ende Update() */
[...]
```

Die Methode *Update()* führt den eigentlichen Update gegen die Datenbank aus. Zuerst wird der SQL-Befehl zusammen gebaut und den Tabellenspalten der neue Wert zugewiesen. Alle Befehlsteile und Variableninhalte werden per String-Concat aneinander gehängt. Der Concat wird technisch ein bißchen schneller verarbeitet, als die Variante, bei der die *$Variablen* direkt im SQL-Befehl stehen (z.B. *$sSelect1 = $sSelect1.' WHERE lot_no > $this->sLot_no ';*).

What's $this-> ?

Um auf ein Element einer Klasse innerhalb der selben Klasse zugreifen zu können, muß als Namen der Instanz *$this* verwendet werden. Deshalb erscheinen beim obigen Update() (Listing 16) die neuen Werte auch als *$this->sOk.* Mehr zu objektorientierter Programmierung und OO-Konzepte erfahren Sie beispielsweise hier [5].

Und was ist mit Binding ?

Wenn ein bestimmter SQL-Befehl häufig eingesetzt wird - z.B. beim Einfügen mehrerer Datensätze – lohnt es sich über Host-Variablen und Binding nachzudenken. Üblicherweise wird *ODBC_EXEC()* eingesetzt, um mit der Datenbank zu verhandeln. *ODBC_EXEC*() besteht allerdings aus zwei Verarbeitungsschritten: *Prepare* und *Execute*. *Prepare* prüft den SQL-Befehl syntaktisch und inhaltlich (Tabelle ok, Spalten ok etc.), während *Execute* den Befehl ausführt. Falls Sie mit einer Schleife mehrere Datensätze hintereinander einfügen möchten, genügt eigentlich *1 mal Prepare* und *x mal Execute.* **Wie?** Anstatt der $Variable wird ein Platzhalter »?« in den SQL-Befehl eingesetzt. **Warum?** Diese Vorgehensweise ist um einiges schneller. Probieren sie es aus, aber leider nicht mit SQLBase. PHP mit getrenntem Prepare und Execute wird von meiner relativ alten SQLBase Version (7.5.1) offensichtlich nicht unterstützt (Listing 17).

```
Listing 17:
[...]
UPDATE sysadm.inspection_data
   SET style_id = ?,
   production_date = ?,
   ok = ?
   WHERE lot_no = '14568'
[...]
```

```
Warning: odbc_execute() [function.odbc-execute]: SQL error: [MERANT][ODBC SQLBase
driver]Inconsistent descriptor information., SQL state S1021 in SQLExecute.
```

Wie es aussieht und funktioniert, habe ich deshalb mit IBM DB2 (v.8.1) geprüft:

```
Listing 18:
[...]
  /**
  * UPDATE mit getrenntem Prepare() und Execute(), um Binding nutzen zu können
  */
  function Update ($p_hConnection = 0,$bPrepare = true) {

   if ($bPrepare == true) {
    $sUpdate1 = '';
    $sUpdate1 = $sUpdate1.'UPDATE tw.inspection_data SET ';
    $sUpdate1 = $sUpdate1.' style_id = ?, ';
    $sUpdate1 = $sUpdate1.' production_date = ?, ';
    $sUpdate1 = $sUpdate1.' ok = ? ';
    $sUpdate1 = $sUpdate1.' WHERE lot_no = ? ';

    /**
    * ODBC_PREPARE ausführen und im Fehlerfall ein ROLLBACK auslösen
    */
    $this->UpdateStmt = odbc_prepare($p_hConnection,$sUpdate1);
    if ($this->UpdateStmt == false) {
     odbc_exec($p_hConnection,'ROLLBACK');
     return false;
    }
```

```
   }

   if ($this->UpdateStmt) {
    /**
    * Array mit den Binding Variablen füllen. Reihenfolge beachten !!
    */
    $aUpdate = array();
    $aUpdate['style_id'] = $this->nStyle_id;
    $aUpdate['production_date'] = $this->sProduction_date;
    $aUpdate['ok'] = $this->sOk;
    $aUpdate['lot_no'] = $this->sLot_no;

    /**
    * ODBC_EXECUTE ausführen und im Fehlerfall ein ROLLBACK auslösen
    */
    if (odbc_execute($this->UpdateStmt,$aUpdate)) {
     /* Update ok */
     return true;
    } else {
     odbc_exec($p_hConnection,'ROLLBACK');
     return false;
    }
   }
  } /* Ende Update() */
[...]
```

Wie bereits gesagt: Binding steigert unter bestimmten Voraussetzungen die Verarbeitungsgeschwindigkeit, aber Hand aufs Herz, wie häufig haben Sie bei entsprechender Gelegenheit dieses sinnvolle Feature im GTD genutzt? Eher werden Performanceprobleme mit schneller Hardware erschlagen. In Tabelle 1, sehen Sie eine kleine Gegenüberstellung der wichtigsten SqlBefehle.

Tabelle 1: kurze Gegenüberstellung der wesentlichen Datenbank – Funktionen

	Gupta Team Developer	PHP ODBC
Prüfen und ausführen	SqlPrepareAndExecute()	odbc_exec()
Prüfen	SqlPrepare()	odbc_prepare()
Ausführen	SqlExecute()	odbc_execute()
Lesen	SqlFetchNext()	odbc_fetch_row() und odbc_result()
Commit	SqlCommit()	odbc_commit()

Um beim Thema Geschwindigkeit zu bleiben. Stored Procedures sind das Pseudonym für schnelle Datenverarbeitung. Dies ist der nächste Testpunkt.

PHP und SQLBase Stored Procedures

Bei Stored Procedures (SP) handelt es sich um kleine Verarbeitungsprogramme, die direkt in der Datenbank hinterlegt werden. Um Sie zu nutzen, müssen im Clientprogramm nur der Prozedurenname und die notwendigen Parameter bzw. Rückgabewerte bekannt sein. Eine komplexe Verarbeitung wird einmalig erstellt und in der SP zentral verwaltet. Detailwissen um den Inhalt ist beim Nutzer (Clientprogramm) nicht notwendig. Bei Oracle zum Beispiel werden Stored Procedures (PL/SQL) massiv verwendet und eingesetzt. Eine einfache Stored Procedures ala SQLBase sieht so aus:

```
Listing 19: (StoredProcedure.sql)

STORE MyCount
PROCEDURE MyCount
 PARAMETERS
  Receive Number: nCount
 LOCAL VARIABLES
  Sql Handle: hSql
  Number: nFetchResult
  String: sQuery
 ACTIONS
```

```
   Set nCount = 0
   Call SqlConnect(hSql)
   Set sQuery = 'SELECT COUNT (*) FROM sysadm.product INTO :nCount '
   If SqlPrepareAndExecute(hSql,sQuery)
    If SqlFetchNext(hSql,nFetchResult)
   Call SqlPrepareAndExecute(hSql,'ROLLBACK')
   Call SqlDisconnect(hSql);
```

Die Stored Procedure mit dem Namen *MyCount* liefert als Rückgabewert die Anzahl der Datensätze der Tabelle *SYSADM.PRODUCT.*

```
Listing 20: (test_sp.php)
[...]
$hCursor = odbc_exec($hSqlA,"{call sysadm.MyCount() }");
 if ($hCursor) {

  if ($Row = odbc_fetch_row($hCursor)) {
   $nCount  = odbc_result($hCursor,1);
   echo "count=".$nCount;
  }
 }
[...]
```

Skript Ausgabe ➔: count=7

Sinnvoller wäre obiges Beispiel, wenn eine Parameterübergabe von *Schema* und *Tabellenname* möglich wäre (MyCount2), aber leider habe ich dafür keine Lösung gefunden. Weder Variante:

```
$hCursor = odbc_prepare($hSqlA,"{call sysadm.MyCount2( ?, ?) }");

Warning: odbc_execute() [function.odbc-execute]: SQL error: [MERANT][ODBC SQLBase
driver]Inconsistent descriptor information., SQL state S1021 in SQLExecute
```

noch:

```
$hCursor = odbc_prepare($hSqlA,"{call sysadm.MyCount2('?','?') }");

Warning: odbc_execute() [function.odbc-execute]: SQL error: [MERANT][ODBC SQLBase
driver][SQLBase]00906 PRS ITN Invalid table name, SQL state 37000 in SQLExecute
```

brachten das erhoffte Ergebnis. Aber wie schon gesagt, eventuell liegt es auch an dem hier verwendeten – relativ alten – SQLBase ODBC-Router.

Zusammenfassung

Abschließend läßt sich behaupten, dass PHP und SQLBase sich - mit einigen Abstrichen – recht gut verstehen. Speziell das eingangs angesprochene Intranet läßt sich gut realisieren. Eine browserbasierte Anzeige bestimmer Auswertungen ist schnell bewerkstelligt. Wenn Sie Ihr zukünftiges PHP Intranet jetzt noch flexibel auf verschiedene Datenbanken ausrichten möchten, lohnt sich frühzeitig ein Blick auf die PHP Abstraktionsklassen [6] und [7], sowie das PHP Template Smarty [8]. Basierend auf diesen OpenSource Werkzeugen, können mächtige Lösungen entstehen.

Links und Literatur

[1] Forum.gupta.marketing (03. Mai 2005) news://newsgroup.guptaworldwide.com
[2] SELFHTML – Stefan Münz http://de.selfhtml.org/
[3] Jens Bohl Diplomarbeit 30.12.2002: „Möglichkeiten der Gestaltung flexible Softwarearchitekturen..“
http://cybop.berlios.de/papers/2003_flexible_presentation_layer_diploma/Diplomarbeit.pdf
[4] SQLBaseHtmlDoc.zip Thomas Wiedmann http://www.twiedmann.de/centura.php
[5] Richard Samar, Christian Stocker: PHP de Luxe , 2002. ISBN 3-8266-0799
[6] Datenbank Abstraktion - ADOdb – http://adodb.sourceforge.net
[7] Datenbank Abstraktion – PEAR:DB – http://pear.php.net/package/DB
[8] Smarty Template – http://smarty.php.net

Abkürzungen und Quellen

- Apache Apache Web-Server (http://www.apache.org)
- CSS Cascading Stylesheets (http://www.w3.org)
- GTD Gupta Team Developer (http://www.guptaworldwide.com)
- HTML Cascading Stylesheets (http://www.w3.org)
- PHP Hypertext Preprocessor (http://www.php.net)