Eine kurze Einführung in PHP und die Oracle Anbindung

geschrieben von Sascha Pfalz

In diesem Dokument soll ein kurzer Überblick über die Scriptsprache PHP und deren Möglichkeiten zur Kommunikation mit Oracle Datenbanken aufgezeigt werden. Ich möchte hier aber kein vollständiges Tutorial oder gar einen kompletten Kurs niederschreiben sondern eher einen allgemeinen Überblick über die Einsatzmöglichkeiten sowie die Vor- und Nachteile der Sprache PHP in Verbindung mit Oracle Datenbanken aufzeigen.

1. Was ist PHP überhaupt, wofür braucht man das?

Bei der Sprache PHP handelt sich es um eine interpretierende Scriptsprache, die in verschiedenen Variationen auf einem (Web-)Server installiert und ausgeführt werden kann. Der grösste Teil der derzeitigen Installationen wird sicherlich als Servermodul laufen, da diese Methode die grösste Performance und auch die höchste Sicherheit im Internet-Umfeld bietet.
Als weitere Methoden der Ausführung des PHP Interpreters gibt es noch ein CGI Executable, ein Commandline Interpreter sowie eine Erweiterung mit GTK+, mit der dann sogar grafische Oberflächen in PHP programmiert werden können.
PHP wird in erster Linie zur Erstellung dynamischer Websites wie z.B. Foren, Portalseiten o.ä. eingesetzt, ist aber auch durchaus für Back-Office Anwendungen oder Shell-Scripting geeignet. Aufgrund der interpretierenden Eigenschaft von PHP ist die Sprache naturgemäß nicht so schnell wie kompilierende Sprachen (z.B. C), jedoch immer noch ausreichend schnell genug um auch komplexe Anwendungen zu realisieren.

Die Syntax von PHP ist eng an C angelehnt, auch der grösste Teil der Funktionsnamen ist analog zu den darunterliegenden C-Bibliotheken aufgebaut, so das sich ein geübter C-Programmierer relativ schnell in die Sprache einarbeiten kann.

Für eine Liste der genauen Details sei auf die Homepage von PHP verwiesen, die unter der Adresse http://www.php.net erreichbar ist.

2. Und wie geht das?

Die Verwendung von PHP ist aufgrund der interpretierenden Eigenschaften der Sprache relativ einfach. Meistens wird der PHP Code direkt in eine HTML Seite engebettet, wobei spezielle Marker das Anfang und Ende des PHP Codes für den PHP Interpreter markieren. Diese Marker sind:

Start: <? oder <?php
Ende: ?>

Ausserdem müssen HTML Seiten mit PHP Code eine spezielle Dateiendung besitzen, damit der Webserver die Seiten auch dem korrekten Parser (in dem Fall PHP) zur Bearbeitung übergeben kann. In der normalen Installation von PHP sind diese Dateiendungen „*.php“, „*.php3“ bzw. „*.phps“ . Natürlich lässt sich dieses Verhalten auch umkonfigurieren das z.B. alle Dateien mit der Endung *.html ebenfalls von PHP geparsed werden. Wie genau die Umgebung konfiguriert ist sollte man den Administrator des zu verwendeten Webservers fragen.

Hier jetzt ein kurzes Beispiel zur Verwendung von PHP innerhalb einer normalen Webseite. Diese könnte dann z.B. unter den Namen helloworld.php abgespeichert werden:

```
<html>
<head>
  <title>Ein Test</title>
</head>
<body>
<?php echo('Hello World');?>
</body>
</html>
```

Dieses Beispiel ist bereits vollständig lauffähig und würde in einem Browser die einfache Ausgabe **Hello World produzieren.**
In dem Beispiel wird die Funktion echo() zur Ausgabe eines Strings (Hello World) verwendet.Eine ausführliche Liste aller verfügbaren Funktionen und Erweiterungen kann über die URL http://www.php.net/manual/de/ abgerufen werden. Das Online Manual von PHP ist in vielen Sprachen übersetzt und ist auch als *.chm Datei (Windows Hilfe) verfügbar.

Da PHP Open-Source ist werden ständig neue Funktionen und Features in die Sprache integriert, es lohnt sich also durchaus, öfters mal die PHP Website zu besuchen.

3. Okay, und wie war das mit Oracle?

PHP bietet Unterstützung für alle verbreiteten Datenbanken, das reicht von MySQL über PostgreSQL bis zu Oracle einschliesslich der Version 10g. Da PHP in C entwickelt wurde sind auch sämtliche Datenbank Anbindungen über die jeweiligen C-Libraries integriert worden, wobei ein Großteil der Funktionsnamen identisch geblieben ist.
Für Oracle bietet PHP zwei unterschiedliche APIs an, zum einen die „normale“ Oracle Schnittstelle, die bis einschliesslich Oracle 7.x verwendet wurde. Ausserdem noch die weitaus leistungsfähigere OCI8 Schnittstelle, die auf der OCI API von Oracle (ich glaube seit Oracle 8i) basiert und Unterstützung für LOBs, Collections und Bind-Variablen bietet.
Ich werde mich in allen weiteren Ausführungen nur auf die OCI8 Schnittstelle beziehen, da diese eine weitaus bessere Anbindung an Oracle ermöglicht.

Die Oracle OCI8 API bietet eigentlich alles, was man benötigt um mit Oracle effizient zu kommunizieren, wobei jedoch ein (in meinen Augen) großer Nachteil nicht ausser Acht gelassen werden darf, namentlich das Problem der „persistent Connections“.

Der normale Ablauf einer PHP-gestützten Website wird wie folgt ablaufen (stark vereinfacht):

- Ein Benutzer ruft eine PHP Seite auf. Der Webserver wird dann anhand der Extension diese Seite dem PHP Parser übergeben, der den Code interpretiert, ausführt und das Ergebnis an die Website zurückreicht.
- Nach dem Ausführen „stirbt“ der PHP Interpreter und mit ihm natürlich auch alle verwendeten Resourcen (z.B. Speicher o.ä.). Das bedeutet aber auch, das jegliche Datenbank Connections nach dem Abarbeiten der PHP Seite beendet werden!
- Daraus folgt logischerweise, das für jeden Aufruf einer PHP Website, die mit einer Datenbank kommunizieren soll, immer ein Connect() durchgeführt werden muss, was bei Oracle im Gegensatz zu z.B. MySQL ein wenig länger dauert und auch die transaktionale Programmierung erschwert, da alle Transaktionen innerhalb einer PHP Seite durchgeführt werden müssen.

Anhand dieses Ablaufs ist es klar, das man mit PHP keine Systeme entwickeln kann, die eine dauerhafte Verbindung zu Oracle ermöglichen, zumindest nicht in einer HTML Umgebung. Als Standalone Script in einer Shell wäre das jedoch möglich, wenn man mit fork() und Threads

arbeiten würde. Allerdings ist der Aufwand der Programmierung natürlich ungleich grösser. Im Webumfeld muss man sich dieser Tatsachen auf alle Fälle bewusst sein und auch dementsprechend das Datenbankmodell bzw. den Ablauf der Software daraufhin abstimmen.

Um die ganzen besprochenen Sachen ein wenig zu verdeutlichen folgt hier mal ein reines PHP Script zum Speichern eines CLOBs in folgender Tabelle:

```
CREATE TABLE CLOBTEST
  (
  ID       NUMBER(38),
  MESSAGE  CLOB
  );
```

Hier der Code (Die Beschreibung dazu folgt nach dem Listing):

```
<?php
$sock = OCILogon("scott","tiger");
if(!$sock)
  {
  die("cannot connect to DB");
  }

// The vars to save:

$mynumber = 1;
$mytext=<<<EOM
This a text which should be added to our CLOB Field.
It can be up to 4GB in size!
EOM;

// The actual query to use:

$query=<<<EOM
INSERT INTO CLOBTEST (ID,MESSAGE) VALUES (:myid,EMPTY_CLOB())
RETURNING MESSAGE INTO :the_clob
EOM;

// Retrieve LOB Locator:

$clob = OCINewDescriptor($sock, OCI_D_LOB);
if(!($stmt = OCIParse($sock,$query)))
  {
  echo("Parse failed.\n");
  OCILogoff($sock);
  exit;
  }

// Bind the vars:

OCIBindByName ($stmt, ":the_clob", $clob, -1, OCI_B_CLOB);
OCIBindByName($stmt, ":myid",$mynumber,-1);

// Now execute the query, but do not commit yet:

if(OCIExecute($stmt, OCI_DEFAULT)==false)
```

```
  {
  echo("Cannot execute!\n");
  OCIFreeStatement($stmt);
  OCILogoff($sock);
  exit;
  }
// Now save the lob contents:

if($clob->save($mytext))
  {
  OCICommit($sock);
  }
else
  {
  echo("Unable to add CLOB text to database!\n");
  OCIRollBack($sock);
  }
OCIFreeDesc($clob);
OCIFreeStatement($stmt);
OCILogoff($sock);
?>
```

Mit diesem Beispiel wird ein beliebiger Text in ein CLOB Feld geschrieben. Hier eine kurze Erklärung zu dem Listing:

Zuerst muss mit OCILogon() eine Verbindung aufgebaut werden. Diese returniert ein Resultset, welches für die weitere Kommunikation benötig wird. Wir connecten hier direkt zur lokalen Datenbank, man kann aber auch einen gültigen TNS Namen als dritten Parameter übergeben. Ist dieser leer oder nicht angegeben wird zur Datenbank ORACLE_SID connected (Environment Variable).

Dann werden Werte in die Variablen $mynumber, $mytext und $myquery geschrieben. Hier sieht man auch, das PHP das Setzen von sogenannten „HERE-Document“ Typen ermöglicht, analog zu Perl.
Der eigentliche Query steht in der Variablen $myquery und verwendet Bind Variablen (:myid und :the_clob).

Da wir ein CLOB Feld bestücken wollen, benötigen wir einen LOB-Locator, um später den Text in $mytext abzuspeichern, das erledigt die Funktion OCINewDescriptor(), die einen LOB Locator returniert.

Als nächstes wird unser Query geparsed mit OCIParse(), daraus wird ein Statement handle generiert und Oracle „pre-compiled“ diesen Query bzw. prüft den SQL-Cache nach bereits vorhandenen Kopien dieses Queries und damit den zu verwendenen Execution-Plan.

Nach dem Parsen des Queries müssen die Bind-Variablen an PHP Variablen gebunden werden, das wird mit der Funktion OCIBindByName() realisiert.

Jetzt kann der Query endlich ausgeführt werden via OCIExecute(). Dabei muss in diesem Fall darauf geachtet werden, das auf JEDEN FALL OCI_DEFAULT als zweiter Parameter übergeben wird, da OCIExecute() per Default (ohne zweiten Parameter) nach einem erfolgreichen Aufruf automatisch ein Commit() ausführt, und das wäre in unserem Fall schlecht, da wir in zwei Schritten das INSERT durchführen müssen und das nach dem Commit nicht mehr könnten.

Hat das Execute geklappt müssen wir noch den Inhalt für unser CLOB Feld abspeichern, das passiert durch Aufruf der Methode save() des returnierten Objects von OCINewDescriptor().

Konnte der Text erfolgreich gespeichert werden muss die Transaktion noch mit einem OCICommit abgeschlossen werden, die allozierten Speicher mit OCIFreeDesc (Lob-Locator) und OCIFreeStatement (kompilierter Query) freigegeben werden und abschliessend die Verbindung zur Datenbank geschlossen werden (OCILogoff).

Dieses etwas komplexere Beispiel zeigt sehr deutlich, wie nah die C-Funktionen der OCI Library auf PHP abgebildet worden sind.
Es existieren noch diverse andere Funktionen zur komfortablen Ansteuerung der diversen Oracle Spezialitäten, hier sei wieder auf die Dokumentation von PHP und der OCI Schnittstelle verwiesen.

Das Listing macht allerdings auch sehr deutlich, das eine Menge PHP Code geschrieben werden muss, um mit OCI zu arbeiten, gerade wenn man Wert auf die Verwendung von Bind-Variablen etc. legt, auch wenn die reine Codemenge im Vergleich zu C auf jeden Fall drastisch kürzer und damit auch übersichtlicher ist. Auf alle Fälle lässt sich mit PHP in sehr kurzer Zeit eine performante Webapplikation in Verbindung mit Oracle aufbauen.

Ausblick – Was noch so kommt

Um das Manko mit dem aufwendigen OCI Code ein wenig zu vereinfachen habe ich eine PHP Klasse entwickelt, mit der die Kommunikation mit Oracle extrem vereinfacht wird. Um z.B. mit Hilfe dieser Klasse einen einfachen Single-Row Query durchzuführen, sind folgende Codezeilen notwenig:

```
<?
include('oci8_class.php');
$db = new db_oci8;
$db->Connect();
$mytime = $db->Query('SELECT SYSDATE FROM DUAL');
$db->Disconnect();
echo('Database time is '.$mytime['SYSDATE']);
?>
```

Dieses kurze Beispiel gibt das SYSDATE der verwendeten Datenbank aus, wobei die Klasse Bind-Variablen ebenfalls unterstützt. Da diese Klasse allerdings derzeit noch intensiv in der Entwicklung ist kann hier an dieser Stelle noch nicht weiter darauf eingegangen werden. Sobald die Klasse alle notwendigen Methoden unterstützt (es fehlt z.B. noch Unterstützung für LOBs/CLOBs und Stored Procedures mit IN/OUT Variablen) werde ich diese Klasse veröffentlichen und auch auf dem DOAG Server verfügbar machen.

Bis dahin kann ich allen Interessierten empfehlen, einfach mal ein paar Tests mit der PHP Schnittstelle durchzuführen. Bei Fragen oder sonstigen Kommentaren zu diesem Tutorial einfach eine E-Mail an webmaster@saschapfalz.de schreiben oder meiner Homepage unter http://www.saschapfalz.de einen Besuch abstatten.

Erfolgreiches Programmieren mit PHP/Oracle.

Sascha Pfalz