PhPepperShop - Performanceanalyse

José Fontanil und Reto Glanzmann

Performanceanalyse der Artikeldarstellung

José Fontanil / Reto Glanzmann Januar 2003

Performanceanalyse der Artikeldarstellung PhPepperShop 1.2

Inhaltsverzeichnis

Zweck dieser Analyse

Nachdem im November 2002 bekannt wurde, dass der Artikeldarstellungsalgorithmus des PhPepperShops nicht mit der Komplexität der Artikel skaliert, musste ein Ausweg erarbeitet werden. In diesem Dokument begründen wir, gestützt auf statistisch erfasste Messwerte, die Änderung des Algorithmus im wohl Performance-sensitivsten Teil des PhPepperShops.

Ziel dieser Performanceanalyse war es, herauszufinden ob der PhPepperShop weiterhin den bisher verwendeten Algorithmus zum Auslesen und Darstellen eines Artikels benutzen soll, oder ob wir den neuen, noch als experimentell eingestuften Algorithmus einsetzen sollen.

Dabei mussten wir die verschiedenen Artikelarten mit Referenzartikeln modellieren und überprüfen, wie die Algorithmen mit den Artikeln zurecht kommen. Bevor wir den neuen Algorithmus einsetzen können, müssen wir 'beweisen' können, dass er auch für ALLE Arten von Artikeln bewährt ist. Nicht, dass wir einen neuen Algorithmus einführen und dann skaliert er dafür mit vielen einfach gestalteten Artikeln nicht mehr.

Lösung

Wie man den Analyseblättern (Seiten 4,5) entnehmen kann, ist der neue Algorithmus (4-SQL Algorithmus) eindeutig der bessere Wurf. Mit wenigen Sekundenbruchteilen Unterschied arbeitet der neue Algorithmus auch mit den bisher vom alten Algorithmus ohne Probleme abgearbeiteten Artikelsorten zufriedenstellend. Bei komplexen Artikeln hingegen ist der neue Algorithmus um Faktor 7 bis Faktor 2000 mal schneller!

Einzig eine Frage muss man sich noch stellen: Wenn der Datenbankserver NICHT – wie bei unseren Tests – auf dem gleichen Server, wie der Webserver steht, so kann es duchaus zu schlechteren Resultaten in allen Artikelkategorien kommen. Wir werden deshalb im PhPepperShop Code vorerst beide Algorithmen drin belassen, werden aber den alten Algorithmus deaktivieren.

Die Algorithmen

Der **'alte' Algorithmus**, hier auch 1-SQL Algorithmus genannt (Datei `USER_ARTIKEL_HANDLING.php`):

```
function getArtikel_deaktiviert($Artikel_ID) {
    global $Database;
    global $sql_getArtikel_1_1;
    global $sql_getArtikel_1_2;
    if (! is_object($Database)) {
        die("<P><H1 class='content'>U_A_H_Error: Datenbank nicht erreichbar</H1></P>\n");
    }
    else {
        //Query ausfuehren und in ResultSet schreiben (Typ ResultSet, siehe database.php)
        $RS = $Database->Query($sql_getArtikel_1_1.trim($Artikel_ID).$sql_getArtikel_1_2);
        $myArtikel = new Artikel; //Ein neues Artikel-Objekt instanziieren
        $grunddaten_eingelesen = 0; // Wird true bei > 0, sobald Artikelgrunddaten EINMAL eingelesen worden sind
        while (is_object($RS) && $RS->NextRow()){
            //Artikel einlesen:
            // Einlesen Teil 1/2: Artikelgrunddaten einlesen
            if (!$grunddaten_eingelesen) {
                $myArtikel->artikel_ID = $RS->GetField("Artikel_ID");
                $myArtikel->artikel_Nr = $RS->GetField("Artikel_Nr");
                $myArtikel->name = $RS->GetField("Name");
                $myArtikel->beschreibung = $RS->GetField("Beschreibung");
                $myArtikel->letzteAenderung = $RS->GetField("letzteAenderung");
                $myArtikel->gewicht = $RS->GetField("Gewicht");
                $myArtikel->preis = $RS->GetField("Preis");
                $myArtikel->aktionspreis = $RS->GetField("Aktionspreis");
                $myArtikel->link = $RS->GetField("Link");
                $myArtikel->bild_gross = $RS->GetField("Bild_gross");
```



```
                $myArtikel->bild_klein = $RS->GetField("Bild_klein");
                $myArtikel->bildtyp = $RS->GetField("Bildtyp");
                $myArtikel->bild_last_modified = $RS->GetField("Bild_last_modified");
                $myArtikel->aktionspreis_verwenden = $RS->GetField("Aktionspreis_verwenden");
                $myArtikel->mwst_satz = $RS->GetField("MwSt_Satz");
                $myArtikel->aktion_von = $RS->GetField("Aktion_von");
                $myArtikel->aktion_bis = $RS->GetField("Aktion_bis");
                $myArtikel->zusatzfelder_text = explode("þ",$RS->GetField("Zusatzfeld_text"));
                $myArtikel->zusatzfelder_param = explode("þ",$RS->GetField("Zusatzfeld_param"));
            }
            $grunddaten_eingelesen++; // Flag wird beim naechsten Durchgang > 0 = false
            // Einlesen Teil 2/2: Optionen, Variationen, Variationsgruppen eines Artikels einlesen
            $myArtikel->putoption($RS->GetField("Optionstext"),$RS->GetField("Preisdifferenz"));
            $myArtikel->putopt_gewicht($RS->GetField("Optionstext"),$RS->GetField("Gewicht_Opt"));
            $array_preis[$RS->GetField("Variations_Nr")] = $RS->GetField("Aufpreis");
            $array_text[$RS->GetField("Variations_Nr")] = $RS->GetField("Variationstext");
            $array_gruppe[$RS->GetField("Variations_Nr")] = $RS->GetField("Variations_Grp");
            $array_gewicht_var[$RS->GetField("Variations_Nr")] = $RS->GetField("Gewicht_Var");
            $myArtikel->var_gruppen_text[$RS->GetField("Gruppen_Nr")] = $RS->GetField("Gruppentext");
            $myArtikel->var_gruppen_darst[$RS->GetField("Gruppen_Nr")] = $RS->GetField("Gruppe_darstellen");
        }//End while
        // Damit die Optionen ihre richtige Position behalten wird dem SQL ein ORDER BY Optionen_Nr Statement
           hinzugefuegt
        // Damit jetzt auch die Variationen ihre Reihenfolge behalten koennen, werden diese sortiert in zwei
           Arrays ausgelesen
        // und dann in einer weiteren Schleife korrekt sortiert in ihr eigentliches Zielarray im Artikel-Objekt
           abgelegt.
        for($i=1;$i <= count($array_preis);$i++) {
            $myArtikel->putvariation($array_text[$i],$array_preis[$i]);
            $myArtikel->putvar_gruppe($array_text[$i],$array_gruppe[$i]);
            $myArtikel->putvar_gewicht($array_text[$i],$array_gewicht_var[$i]);
        } // end of for
        return $myArtikel;
    }//End else
}//End getArtikel (momentan deaktiviert)
```

Verwendete SQLs:

```
$sql_getArtikel_1_1 = '
SELECT a.*,ao.*,av.*,ag.*
FROM (artikel a LEFT JOIN artikel_optionen ao ON a.Artikel_ID = ao.FK_Artikel_ID)
     LEFT JOIN artikel_variationen av ON a.Artikel_ID = av.FK_Artikel_ID
     LEFT JOIN artikel_variationsgruppen ag ON a.Artikel_ID = ag.FK_Artikel_ID
WHERE a.Artikel_ID = "';

$sql_getArtikel_1_2 = '" ORDER BY ao.Optionen_Nr';
```

Der in v.1.03 entstandene und in v.1.05 bis v.1.2 erweiterte Algorithmus wurde daraufhin optimiert möglichst nur einmal mit der Datenbank Verbindung aufnehmen zu müssen. Weil wir davon ausgingen, dass die Datenbankverbindung resp. deren Aufbau das zeitintensivste Element einer Artikeldarstellung sei.

Bei komplexen Artikeln wie z.B. dem Artikel Bolt (sieben Variationsgruppen mit total 58 Varianten und 15 Optionen) ergibt sich aber durch die Left-Joins eine ungeheuer grosse Anzahl an abzuarbeitenden Zeilen. Dies macht dann das Abfüllen der SQL-Daten in das Artikelobjekt zum mitunter sehr zeitintensiven Flaschenhals. Das Darstellen EINES Bolt Artikels braucht ~ 12 Sekunden mit obigem Algorithmus.

Der **'neue' Algorithmus**, hier auch 4-SQL Algorithmus genannt (Datei USER_ARTIKEL_HANDLING.php):

```
function getArtikel($Artikel_ID) {
    global $Database;
    global $sql_getArtikel_1_3;
    global $sql_getArtikel_1_4;
    global $sql_getArtikel_1_5;
    global $sql_getArtikel_1_6;
    global $sql_getArtikel_1_7;

    if (! is_object($Database)) {
        die("<P><H1 class='content'>U_A_H_Error: Datenbank nicht erreichbar</H1></P>\n");
    }
    else {
        // Query ausfuehren und in ResultSet schreiben (Typ ResultSet, siehe database.php)
        // Artikelgrunddaten auslesen
        $RS = $Database->Query($sql_getArtikel_1_3.trim($Artikel_ID));
        $myArtikel = new Artikel; //Ein neues Artikel-Objekt instanziieren
        while (is_object($RS) && $RS->NextRow()){
            $myArtikel->artikel_ID = $RS->GetField("Artikel_ID");
            $myArtikel->artikel_Nr = $RS->GetField("Artikel_Nr");
            $myArtikel->name = $RS->GetField("Name");
            $myArtikel->beschreibung = $RS->GetField("Beschreibung");
            $myArtikel->letzteAenderung = $RS->GetField("letzteAenderung");
            $myArtikel->gewicht = $RS->GetField("Gewicht");
            $myArtikel->preis = $RS->GetField("Preis");
```



```
            $myArtikel->aktionspreis = $RS->GetField("Aktionspreis");
            $myArtikel->link = $RS->GetField("Link");
            $myArtikel->bild_gross = $RS->GetField("Bild_gross");
            $myArtikel->bild_klein = $RS->GetField("Bild_klein");
            $myArtikel->bildtyp = $RS->GetField("Bildtyp");
            $myArtikel->bild_last_modified = $RS->GetField("Bild_last_modified");
            $myArtikel->aktionspreis_verwenden = $RS->GetField("Aktionspreis_verwenden");
            $myArtikel->mwst_satz = $RS->GetField("MwSt_Satz");
            $myArtikel->aktion_von = $RS->GetField("Aktion_von");
            $myArtikel->aktion_bis = $RS->GetField("Aktion_bis");
            $myArtikel->zusatzfelder_text = explode("þ",$RS->GetField("Zusatzfeld_text"));
            $myArtikel->zusatzfelder_param = explode("þ",$RS->GetField("Zusatzfeld_param"));
        }//End while

        // Query ausfuehren und in ResultSet schreiben (Typ ResultSet, siehe database.php)
        // Artikeloptionen auslesen
        $RS = $Database->Query($sql_getArtikel_1_4.trim($Artikel_ID).$sql_getArtikel_1_5);
        while (is_object($RS) && $RS->NextRow()){
            $myArtikel->putoption($RS->GetField("Optionstext"),$RS->GetField("Preisdifferenz"));
            $myArtikel->putopt_gewicht($RS->GetField("Optionstext"),$RS->GetField("Gewicht_Opt"));
        }//End while

        // Query ausfuehren und in ResultSet schreiben (Typ ResultSet, siehe database.php)
        // Artikelvariationen auslesen
        $RS = $Database->Query($sql_getArtikel_1_6.trim($Artikel_ID));
        while (is_object($RS) && $RS->NextRow()){
            $array_preis[$RS->GetField("Variations_Nr")] = $RS->GetField("Aufpreis");
            $array_text[$RS->GetField("Variations_Nr")] = $RS->GetField("Variationstext");
            $array_gruppe[$RS->GetField("Variations_Nr")] = $RS->GetField("Variations_Grp");
            $array_gewicht_var[$RS->GetField("Variations_Nr")] = $RS->GetField("Gewicht_Var");
        }//End while
        // Query ausfuehren und in ResultSet schreiben (Typ ResultSet, siehe database.php)
        // Variationsgruppen auslesen
        $RS = $Database->Query($sql_getArtikel_1_7.trim($Artikel_ID));
        while (is_object($RS) && $RS->NextRow()){
            $myArtikel->var_gruppen_text[$RS->GetField("Gruppen_Nr")] = $RS->GetField("Gruppentext");
            $myArtikel->var_gruppen_darst[$RS->GetField("Gruppen_Nr")] = $RS->GetField("Gruppe_darstellen");
        }//End while

        // Damit die Optionen ihre richtige Position behalten wird dem SQL ein ORDER BY Optionen_Nr Statement
           hinzugefuegt
        // Damit jetzt auch die Variationen ihre Reihenfolge behalten koennen, werden diese sortiert in zwei
           Arrays ausgelesen
        // und dann in einer weiteren Schleife korrekt sortiert in ihr eigentliches Zielarray im Artikel-Objekt
           abgelegt.
        for($i=1;$i <= count($array_preis);$i++) {
            $myArtikel->putvariation($array_text[$i],$array_preis[$i]);
            $myArtikel->putvar_gruppe($array_text[$i],$array_gruppe[$i]);
            $myArtikel->putvar_gewicht($array_text[$i],$array_gewicht_var[$i]);
        } // end of for
        return $myArtikel;
    }//End else
}//End getArtikel (optimiert)
```

Verwendete SQLs:

```
$sql_getArtikel_1_3 = '
SELECT a.*
FROM artikel a
WHERE a.Artikel_ID = ';

$sql_getArtikel_1_4 = '
SELECT ao.*
FROM artikel_optionen ao
WHERE ao.FK_Artikel_ID = ';

$sql_getArtikel_1_5 = ' ORDER BY ao.Optionen_Nr';

$sql_getArtikel_1_6 = '
SELECT av.*
FROM artikel_variationen av
WHERE av.FK_Artikel_ID = ';

$sql_getArtikel_1_7 = '
SELECT ag.*
FROM artikel_variationsgruppen ag
WHERE ag.FK_Artikel_ID = ';
```

Beim neuen Algorithmus war es das Ziel die durch SQL-Joins erstellten 'Dubletteneinträge' auf Null zu reduzieren. Wir haben deshalb alle Joins in einzelne SQL-Abfragen aufgesplittet und die Verarbeitung des Objektabfüllens dementsprechend in der Funktion angepasst. In der Shopversion v.1.3 konnten wir das Objekthandling noch etwas weiter beschleunigen.

Performanceanalyse IDgetArtikeleinerKategorie[vonbis] Algorithmen

Testbrowser Microsoft© Internet Explorer 6.0

Test 1: Nach dem Funktionsaufruf IDgetArtikeleinerKategorie[vonbis]
Test 2: Zeitbedarf der Anzeige von NACH obigem Funktionsaufruf bis zum vollständigen abarbeiten des Codes in darstellen = 1, body-Teil

Zeitangaben erfolgen in Sekunden (Zeiterfassung mittels PHP-Funktion microtime( ))

Szenario 1:

Kategorie DVDs: 39 Artikel, von welchen keiner Optionen/Variationen/Variationsgruppen/Kundeneingabe-texte enthaelt.
Zweck: Dieser Test steht für viele Artikel in einer Kategorie, welche wenig Komplexität aufweisen.

	IDgetArtikeleinerKategorie								IDgetArtikeleinerKategorievonbis							
	4 SQL-Algorithmus				***1 SQL-Algorithmus***				***4 SQL-Algorithmus***				***1 SQL-Algorithmus***			
Test 1	0.405	0.301	0.314	0.323	0.411	0.281	0.274	0.297	0.478	0.379	0.366	0.368	0.489	0.394	0.405	0.407
Test 2	0.113	0.133	0.165	0.158	0.138	0.177	0.174	0.181	0.258	0.135	0.283	0.188	0.136	0.190	0.169	0.173
Totalzeitbedarf	0.518	0.434	0.479	0.481	0.549	0.458	0.448	0.478	0.736	0.514	0.649	0.556	0.625	0.584	0.574	0.580
Durchschnitt	0.478				0.483				0.614				0.591			
Mozilla v.1.1	0.615				0.585				0.732				0.804			

Szenario 2:

Eine Kategorie mit nur einem Artikel drin, welcher nur eine Option und zwei Variationen hat.
Zweck: Dieser Test überprüft den Zeitbedarf der Artikeldarstellung mit einem Referenzartikel.

	IDgetArtikeleinerKategorie								IDgetArtikeleinerKategorievonbis							
	4 SQL-Algorithmus				***1 SQL-Algorithmus***				***4 SQL-Algorithmus***				***1 SQL-Algorithmus***			
Test 1	0.017	0.014	0.015	0.014	0.019	0.023	0.018	0.019	0.044	0.024	0.024	0.023	0.025	0.032	0.026	0.023
Test 2	0.058	0.034	0.013	0.013	*0.160*	0.018	0.016	0.021	*0.150*	0.016	0.026	0.014	*0.155*	0.022	0.016	0.013
Totalzeitbedarf	0.075	0.048	0.028	0.027	0.179	0.041	0.034	0.040	0.194	0.040	0.050	0.037	0.180	0.054	0.042	0.036
Durchschnitt	0.045				0.074				0.080				0.078			
Mozilla v.1.1	0.032				0.079				0.065				0.039			

Szenario 3:

Eine Kategorie mit einem sehr komplexen Artikel drin (Kategorie Gitarren->Bolt, Artikel Bolt). Der Artikel Bolt hat neben Namen, Artikelnummer und Beschreibung (mittlere Länge) auch noch sieben Variations-gruppen mit total 58 Varianten und 15 Optionen, davon sind 14 Aufpreisbehaftet. Weiter hat dieser Artikel noch ein Kundentexteingabefeld.
Zweck: Dieser Test überprüft den Zeitbedarf der Artikeldarstellung mit einem sehr komplexen Artikel.

	IDgetArtikeleinerKategorie								IDgetArtikeleinerKategorievonbis							
	4 SQL-Algorithmus				***1 SQL-Algorithmus***				***4 SQL-Algorithmus***				***1 SQL-Algorithmus***			
Test 1	0.215	0.190	0.235	0.185	11.546	11.526	11.662	11.495	2.842	2.681	3.010	2.927	12.956	12.987	12.999	12.954
Test 2	0.118	0.132	0.079	0.147	0.086	0.093	0.096	0.092	0.089	0.077	0.100	0.101	0.077	0.088	0.089	0.081
Totalzeitbedarf	0.333	0.322	0.314	0.332	11.632	11.619	11.758	11.587	2.931	2.758	3.110	3.028	13.033	13.075	13.088	13.035
Durchschnitt	0.325				11.649				2.957				13.058			
Mozilla v.1.1	0.256				11.996				3.086				13.576			

Testumgebung: Athlon 1GHz, 512MB RAM, Apache 1.3.22 auf XP, PHP 4.1.1 (kein Encoder, Accelerator)
Datum: 11. November 2002, *Version*: PhPepperShop v.1.2 inkl. bis dato bekannten Bugfixes eingespielt

Performanceanalyse IDgetArtikeleinerKategorie[vonbis] Algorithmen

Testbrowser Mozilla v.1.1 (www.mozilla.org)

Test 1: Nach dem Funktionsaufruf IDgetArtikeleinerKategorie[vonbis]
Test 2: Zeitbedarf der Anzeige von NACH obigem Funktionsaufruf bis zum vollständigen abarbeiten des Codes in darstellen = 1, body-Teil

Zeitangaben erfolgen in Sekunden (Zeiterfassung mittels PHP-Funktion microtime( )) grün = schneller

Szenario 1:

Kategorie DVDs: 39 Artikel, von welchen keiner Optionen/Variationen/Variationsgruppen/Kundeneingabetexte enthaelt.
Zweck: Dieser Test steht für viele Artikel in einer Kategorie, welche wenig Komplexität aufweisen.

	IDgetArtikeleinerKategorie								IDgetArtikeleinerKategorievonbis							
	4 SQL-Algorithmus				***1 SQL-Algorithmus***				***4 SQL-Algorithmus***				***1 SQL-Algorithmus***			
Test 1	0.368	0.282	0.300	0.299	0.392	0.268	0.284	0.279	0.618	0.353	0.362	0.365	0.524	0.385	0.398	0.407
Test 2	0.151	0.284	0.384	0.390	0.145	0.292	0.327	0.353	0.159	0.373	0.420	0.279	0.164	0.355	0.472	0.512
Totalzeitbedarf	0.519	0.566	0.684	0.689	0.537	0.560	0.611	0.632	0.777	0.726	0.782	0.644	0.688	0.740	0.870	0.919
Durchschnitt	0.615				0.585				0.732				0.804			
Microsoft IE 6.0	0.478				0.483				0.614				0.591			

Szenario 2:

Eine Kategorie mit nur einem Artikel drin, welcher nur eine Option und zwei Variationen hat.
Zweck: Dieser Test überprüft den Zeitbedarf der Artikeldarstellung mit einem Referenzartikel.

	IDgetArtikeleinerKategorie								IDgetArtikeleinerKategorievonbis							
	4 SQL-Algorithmus				***1 SQL-Algorithmus***				***4 SQL-Algorithmus***				***1 SQL-Algorithmus***			
Test 1	0.016	0.016	0.017	0.014	0.019	0.035	0.031	0.017	0.068	0.023	0.023	0.044	0.022	0.020	0.020	0.022
Test 2	0.023	0.017	0.013	0.013	*0.166*	0.014	0.017	0.017	0.045	0.018	0.025	0.014	0.026	0.017	0.015	0.013
Totalzeitbedarf	0.039	0.033	0.030	0.027	0.185	0.049	0.048	0.034	0.113	0.041	0.048	0.058	0.048	0.037	0.035	0.035
Durchschnitt	0.032				0.079				0.065				0.039			
Microsoft IE 6.0	0.045				0.074				0.080				0.078			

Szenario 3:

Eine Kategorie mit einem sehr komplexen Artikel drin (Kategorie Gitarren->Bolt, Artikel Bolt). Der Artikel Bolt hat neben Namen, Artikelnummer und Beschreibung (mittlere Länge) auch noch sieben Variationsgruppen mit total 58 Varianten und 15 Optionen, davon sind 14 Aufpreisbehaftet. Weiter hat dieser Artikel noch ein Kundentexteingabefeld.
Zweck: Dieser Test überprüft den Zeitbedarf der Artikeldarstellung mit einem sehr komplexen Artikel.

	IDgetArtikeleinerKategorie								IDgetArtikeleinerKategorievonbis							
	4 SQL-Algorithmus				***1 SQL-Algorithmus***				***4 SQL-Algorithmus***				***1 SQL-Algorithmus***			
Test 1	0.131	0.102	0.098	0.172	11.929	11.773	11.955	11.583	2.819	3.083	3.033	2.908	13.179	13.521	13.552	13.389
Test 2	0.226	0.104	0.094	0.098	0.189	0.249	0.102	0.204	0.099	0.210	0.094	0.096	0.209	0.243	0.102	0.110
Totalzeitbedarf	0.357	0.206	0.192	0.270	12.118	12.022	12.057	11.787	2.918	3.293	3.127	3.004	13.388	13.764	13.654	13.499
Durchschnitt	0.256				11.996				3.086				13.576			
Microsoft IE 6.0	0.325				11.649				2.957				13.058			

Testumgebung: Athlon 1GHz, 512MB RAM, Apache 1.3.22 auf XP, PHP 4.1.1 (kein Encoder, Accelerator)
Datum: 11. November 2002, *Version*: PhPepperShop v.1.2 inkl. bis dato bekannten Bugfixes eingespielt