Herbert Braun, Noud van Kruysbergen

Zelf zoeken

Programmeer een zoekmachine voor je website

De navigatie van je website kan nog zo goed in elkaar zitten; soms is het gewoon handiger om een zoekmachine te hebben. Met de juiste zoektermen krijg je dan snel en gesorteerd de bijbehorende resultaten terug. Zo'n mini-Google voor je website kun je zelf maken, zodat je die kunt aanpassen aan je eigen wensen. Zelfs voor beginnende PHP-programmeurs is dat geen probleem.

Het is niet doenlijk om bij het maken van een website iedereen tevreden te stellen. Ook al bouw je de navigatiestructuur nog zo 'logisch' op, een willekeurige bezoeker die specifieke informatie zoekt kan daar heel anders over denken. En ook met een 'fulltext search', zoals een Google-achtige zoekfunctie ook wel heet, heb je geen vervanging voor een navigatiestructuur. Toch kan zo'n zoekfunctie je bezoekers helpen om snel informatie te vinden. Iedere internetgebruiker is immers gewend aan een invoerveld voor zoektermen en een gesorteerd resultatenoverzicht.

Als je voor je website een contentmanagementsysteem gebruikt, heeft dat normaal gesproken al een fulltext zoekfunctie aan boord. Ook als je een website met statische pagina's hebt, zijn er genoeg manieren om daar zelf een bestaande zoekmachine in te integreren. Een voorbeeld is Google's 'Aangepast zoeken' (zie softlink). Yahoo heeft er Search Monkey voor, evenals de krachtige Search BOSS (Build your Own Search Service).

Er zijn ook kant-en-klare oplossingen in PHP of andere programmeertalen, die je op je eigen server kunt installeren. Een voorbeeld is de gratis basisversie van

iSearch. Het opensourceproject Sphider kan bovendien dode links opsporen. Als je er als webmaster wat meer van wilt maken en beheerdersrechten op de server hebt, kun je bijvoorbeeld MnoGoSearch nemen, Sphinx Search of het Zend-framework, dat zijn zoekframework Lucene heeft afgeleid van de oorspronkelijke Java-versie. Voor een webserver waar Perl op draait kun je met Perlfect Search uit de voeten.

Elk van deze oplossingen heeft zijn eigen voordelen, waardoor het misschien overbodig lijkt om helemaal zelf een zoekmachine in elkaar te zetten. Toch zijn er nog genoeg redenen om dat wel te doen. Zo kun je het indexeren, wegen en de weergave van de zoekresultaten helemaal aan je eigen wensen aanpassen. Zoveel extra werk is het niet, want ook aan kant-en-klare software ben je tijd kwijt om deze specifiek te installeren en configureren. En last but not least is het gewoon leuk om zelf een krachtige webapplicatie in elkaar te zetten.

Conceptversie

Het principe van een zoekmachine is simpel: je laat alle HTML-pagina's doorlezen en controleert met een stringvergelijking of een zoekterm (het 'zoekwoord') erin voorkomt. Maar dat is de simpelste aanpak, die je problemen gaat opleveren.

In het gunstigste geval reageert de zoekmachine traag, afhankelijk ervan hoeveel HTML-pagina's hij telkens opnieuw moet doorzoeken voor dat ene woord. Maar dan reageert hij tenminste nog. In het slechtste geval krijg je namelijk een servertime-out. Beter kun je voor het zoeken op de een of andere manier een soort index gebruiken – het liefst in de vorm van een database, zodat hij snel toegankelijk is.

Je hebt dus in ieder geval twee applicaties nodig: één voor het opbouwen van een index en één die vervolgens de index voor het zoeken gebruikt. Een klein indexeerscript in PHP is niet echt een volwaardige vervanging voor een Google-crawler, maar je kunt er wel wat eenvoudige dingen mee doen. Zo moet het script pure tekst uit de pagina's filteren, zonder zich in de tekensets te verslikken.

Het indexeerscript moet bovendien HTML-tags uit de tekst filteren, zodat de zoekresultaten niet worden beïnvloed door de oorspronkelijke opmaak. Eventueel kun je een lijst van stopwoorden gebruiken, waarmee je alle lidwoorden en koppelwoorden uit de tekst kunt halen. Een grondige analyse van de website zit er dan wel niet in, maar het indexeerscript kan in ieder geval herkennen of een zoekterm in een titel of in een link staat – en hoe vaak dat woord eigenlijk voorkomt op de pagina.

In het ideale geval staat de index in een relationele database, die snel en makkelijk toegankelijk is. In principe kun je dezelfde werkwijze ook toepassen op tekstbestanden, waar je normaliter misschien het PHP-commando strpos() zou gebruiken om te kijken of een zoekterm voorkomt. Dat commando werkt echter aanzienlijk langzamer dan het zoeken met een index.

Voor projecten als deze – en eigenlijk voor alle fatsoenlijke webprojecten – is het aan te raden een lokale serveromgeving te creëren en de scripts eerst daar te testen. Anders moet je de bestanden bij elke kleine wijziging naar de server ftp'en om je project te testen. Installeer bijvoorbeeld lokaal XAMPP, een totaalpakket waarmee je o.a. de Apache-webserver en de MySQL-database tegelijk installeert. Ook dit kun je via de softlink vinden, evenals het complete indexeerscript.

De database aanmaken

Het vertrekpunt van het programmeerwerk is de database. Die zullen we eerst moeten hebben. Bij veel goedkope shared-hostingaanbieders krijg je toegang tot één of meerdere MySQL-databases met een paar megabyte opslagruimte. Voor het instellen en het beheren ervan hebben de meeste providers phpMyAdmin op de webserver geïnstalleerd, die je vanuit een browser kunt bedienen. Hiermee kun je een database in elkaar zetten, maar ook SQL-query's uitvoeren. De Structured Query Language (SQL), die zijn oorsprong in de jaren zeventig heeft, is de basistaal waarmee een database wordt geïnstrueerd wat hij moet doen.

Ook phpMyAdmin zit standaard in XAMPP en is te bereiken via http://localhost/phpMyAdmin. Je kunt de database ook bereiken via de console, om lokaal te ontwikkelen en testen. Als je de Opdrachtprompt van Windows gestart hebt, gaat dat zo:

```
[XAMPP-pad]\mysql\bin\mysql.exe -u root
```

Met het volgende SQL-commando maak je een nieuwe database aan met de naam 'phinx':

```
CREATE DATABASE phinx;
```

Met USE phinx; zeg je vervolgens dat je deze database in je huidige sessie wil gaan gebruiken. Het bovenstaande werkt overigens vaak niet bij shared-hostingaanbieders. Doorgaans staan deze providers erop dat ze zelf een database voor je aanmaken, met een bepaalde naam, inlognaam, wachtwoord, en een lijst van hosts waarvandaan verbinding met de database gemaakt mag worden. Ook stellen ze vaak een maximum aan het aantal databases. In noodgevallen kun je de databasetabellen die je nodig hebt voor het indexeren ook in een bestaande database kwijt. Het is dan wel slim om die tabellen een eigen prefix te geven, bijvoorbeeld 'phinx_paginas' in plaats van 'paginas'.

De database heeft een tabel nodig voor de geïndexeerde pagina's en een tabel voor de woorden (potentiële zoektermen). Omdat iedere pagina een aantal woorden heeft en woorden op meerdere pagina's kunnen voorkomen (n:m-relatie), heb je ook een kruistabel nodig om die aan elkaar te koppelen. De paginatabel heeft de url, de titel en het indexnummer nodig:

```
CREATE TABLE paginas (id INT
AUTO_INCREMENT, url VARCHAR(200),
titel VARCHAR(200), PRIMARY KEY(id),
UNIQUE(url));
```

Het datatype INT betekent dat een kolom (hier id) een getal is, VARCHAR duidt op een eindige tekst (in dit geval is die maximaal 200 tekens lang) en de optie AUTO_INCREMENT geeft aan dat de id voor ieder nieuw record in de tabel automatisch verhoogd wordt. PRIMARY KEY definieert de kolom met id als primaire sleutel. Dat betekent dat het id-nummer aanwezig en uniek moet zijn. Het feit dat de url UNIQUE moet zijn, zorgt ervoor dat verkeerde scripts de database niet ruïneren. MySQL zal namelijk verhinderen dat een url tweemaal wordt toegevoegd. De woordentabel ziet er zo uit:

```
CREATE TABLE woorden (id INT AUTO_
INCREMENT, woord VARCHAR(50), PRIMARY
KEY(id), UNIQUE(woord));
```

De derde tabel krijgt de naam frequentie. Deze tabel bestaat uit twee kolommen die gerelateerd zijn aan de id-nummers van de andere twee tabellen. Omdat de hits bij het indexeren ook nog een gewicht moeten krijgen, komt daar als derde de kolom score bij:

```
CREATE TABLE frequentie (pagina_id
INT, woord_id INT, score INT, PRIMARY
KEY(pagina_id, woord_id));
```

De primaire sleutel geldt in dit geval voor twee kolommen, zodat iedere combinatie van pagina en woord slechts één keer voor mag komen.

Het eerste script

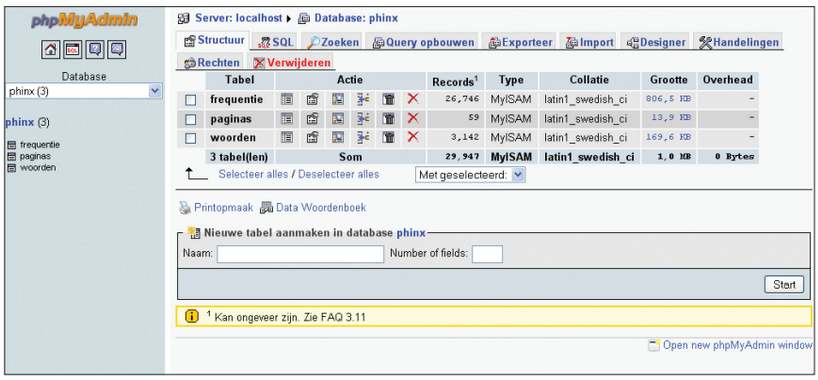
Om de database te vullen, zou het leuk zijn als we net als een crawler van een zoekmachine van link naar link zouden kunnen springen. In de eerste versie van ons indexeerscript maken we het ons echter wat makkelijker. We kijken alleen naar één pagina, waarvan het adres als variabele wordt meegegeven.

PHP-functies als fopen() kunnen via HTTP toegang tot een webpagina krijgen. Bij een standaardinstallatie is deze mogelijkheid vanuit veiligheidsoverwegingen echter vaak geblokkeerd. Aangezien het script sowieso op de server moet draaien en de mappen op internet normaal gesproken dezelfde naam hebben als in het bestandssysteem, accepteert het script gewoon het relatieve lokale pad. Deze methode werkt alleen bij statische websites; de resultaten van server-sidescripts zijn op deze manier niet te indexeren.

```
<?php
$pad = $_GET['pad'] or die('Geen pad
                                meegegeven');
$pad = preg_replace('#^[./]+#', './', $pad);
if (strpos($pad, '/..') !== false) die
                                ('Ongeldig pad!');
if (!get_magic_quotes_gpc()) $pad =
                                addslashes($pad);
...
?>
```

Als je het bovenstaande aanroept als phinx.php?pad=pagina.html, zet de eerste scriptregel de url-parameter in de variabele $pad. Als er geen pad is meegegeven, breekt het script af. De tweede en derde regel zijn bedoeld als een poging om verkeerde invoergegevens nog enigszins te corrigeren.

Allereerst vervangt een reguliere expressie (de uitdrukking achter preg_replace) alle voorafgaande punten en slashes door een './'. Hierdoor zijn alleen mappen onder de huidige map toegestaan. De reguliere expressie ziet er wat raar



Het alles-in-een webserverpakket XAMPP bevat ook phpMyAdmin, het populairste grafische front-end voor MySQL-databases.

uit, omdat er #-tekens staan waar normaal een schuine streep staat, om de expressie af te bakenen. Omdat we zoeken op een schuine streep, is dat korter. Anders zouden we de schuine streep waar we op zoeken moeten laten voorafgaan ('escapen') door een backslash en krijg je /^[.\/]+/. Door het ^-teken zoekt de expressie alleen aan het begin van het pad. Als de map met de pagina's ergens anders staat, dan moet je het tweede argument van preg_replace() daarop aanpassen, bijvoorbeeld '../html/'.

De derde scriptregel verhindert dat de pagina's in een hoger gelegen map kunnen staan. strpos() levert in dat geval 0 of meer terug – anders krijg je een false. In deze regel gebruiken we de PHP-operator !== in plaats van !=, omdat bij !== ook naar het type van het resultaat wordt gekeken. Bij een normale != geldt het getal 0 namelijk ook als false (zie ook het onderscheid tussen == en ===).

Potentieel gevaarlijke tekens zoals aanhalingstekens, vishaken, backslashes en null-bytes worden onschadelijk gemaakt door addslashes(). Dat werkt echter alleen als het standaard geactiveerde magic_quotes_gpc() dat niet al bij het inlezen van alle get-, post- en cookie-variabelen heeft gedaan. Daarom controleren we eerst of deze instelling actief is.

Ondanks deze rudimentaire voorzorgsmaatregelen, mag de indexeerfunctie niet in verkeerde handen vallen. Al is het maar omdat die de server zwaar belast. Het beste is om de functie in een map zetten waarvan de toegang door .htaccess wordt bewaakt. De meeste providers hebben daar wel een tool voor in hun back-end. Of je vraagt meteen aan het begin van het script om een wachtwoord:

```
if (!$_GET['wachtwoord'] == 'totaalgeheim')
                        die('Mooi niet!');
```

PHP heeft een heel scala aan functies om bestanden te openen. Met name file_get_contents() is daarbij erg handig, omdat je daarmee in één keer een heel bestand als string inleest.

```
$inhoud = @ file_get_contents($pad) or
die('Bestand ' . $pad . ' kan niet geopend
worden');
```

Het @-teken zorgt ervoor dat PHP geen waarschuwing geeft als een bestand niet bestaat – dat probleem wordt door het die()-commando afgehandeld.

Op naar de database

Nu is het de hoogste tijd dat ons PHP-script met de database gaat praten:

```
mysql_connect('localhost','root','')
                    or die('Geen verbinding');
mysql_select_db('phinx')
                    or die ('Geen database');
```

De eerste regel maakt verbinding met de MySQL-server, terwijl de tweede regel de juiste database selecteert. De drie argumenten van mysql_connect() zijn achtereenvolgens de host waarop MySQL draait, en de gebruikersnaam en het wachtwoord om op de database in te loggen. In dit geval gebruiken we de standaard databasegebruiker van XAMPP, maar in een productieomgeving mag je om veiligheidsredenen natuurlijk niet als root inloggen op de database. In plaats daarvan krijg je een eigen gebruikersnaam en wachtwoord.

De MySQL-module is nog steeds het populairst om vanuit PHP een database mee te benaderen. Sinds PHP 5 is er echter de opvolger MySQLi, waarbij de i staat voor 'improved'. Hiermee kun je ook een objectgeoriënteerde schrijfwijze gebruiken, evenals 'prepared statements'. Daarmee geef je van tevoren aan wat er staat te gebeuren, waardoor variabelen sneller kunnen worden verwerkt. Het levert ook extra veiligheid op. Bij de oude MySQL-module worden query's namelijk helemaal als PHP-string opgebouwd, waardoor bij minder nette code de kans bestaat dat een aanvaller SQL-code kan binnensluizen (SQL-injectie). Desondanks gebruiken we in ons voorbeeld de MySQL-module, omdat je die makkelijker kunt beheren en deze compatibel is met het nog vaak gebruikte, maar officieel zelfs niet eens meer ondersteunde PHP4.

Om later ook andere scripts op deze manier verbinding te laten maken met de database, kun je bovenstaande regels ook in een eigen scriptbestand zetten, bijvoorbeeld db_verbinding.php. Dit bestand roep je in het indexeerscript met het volgende commando aan:

```
require('db_verbinding.php');
```

Vervolgens voeg je de te indexeren pagina aan de database toe. Het kan zijn dat de pagina al voorkomt, wat je dus beter eerst kunt controleren:

```
$resultaat = mysql_query('SELECT id FROM
          paginas WHERE url = "' . $pad . '"');
$record = mysql_fetch_object($resultaat);
if (!($pagina_id = $record->id)) {
    mysql_query('INSERT INTO paginas (url)
VALUES ("' . $pad . '")') or die('Toevoegen
mislukt');
    $pagina_id = mysql_insert_id();
}
```

Het SQL-commando zoekt in de tabel paginas naar de id van het record dat het meegegeven pad bevat, dus de url die je wilt toevoegen. De opdrachten van je PHP-script aan een MySQL-database instrueer je via mysql_query(). Daarbij is $resultaat een handle om met een aantal fetch()-functies alle records van de resultaattabel een voor een in te lezen. Een voorbeeld daarvan is mysql_fetch_object(), die het gevonden record (dat kan er in dit geval maar één zijn) als object teruglevert.

Als er een record gevonden wordt, krijgt $pagina_id de waarde van $record->id. Als er niets gevonden wordt, heeft $record->id de waarde false en stuurt het script een commando naar de database om de pagina in de database te zetten. Het veld url krijgt dan de waarde van het meegegeven $pad en de database voegt zelf een id toe. Met mysql_insert_id() vraag je vervolgens naar het id-nummer dat de nieuwe pagina heeft gekregen.

Slim HTML crawlen

Voordat je de pagina indexeert, moet het script eerst van de gebruikelijke HTML-overhead worden ontdaan. Let daarbij goed op de tekenset. MySQL bewaart de gegevens standaard namelijk in de codering ISO 8859-1. Dat is voor normale Nederlandse teksten geen probleem, maar internationaal wordt bij websites steeds meer UTF-8 gebruikt. Bovendien wordt deze codering door het kwaliteitsmodel Webrichtlijnen van de Nederlandse overheid aanbevolen. Om eventuele problemen met trema's en umlauten te voorkomen, kun je de tekst dus beter naar UTF-8 converteren:

```
if (stristr($inhoud, 'charset=UTF-8') ||
  preg_match('/encoding=[\'"]UTF-8[\'"]/i',7
                                  $inhoud))
  $inhoud = utf8_decode ($inhoud);
```

De stristr()-functie zoekt naar een UTF-8-declaratie in een <meta>-tag. De reguliere expressie gaat op zoek naar een XML-header (bijvoorbeeld <?xml version="1.0" encoding="UTF-8"?>). De i in stristr() en de /i zorgen ervoor dat deze commando's geen rekening houden met hoofd- of kleine letters.

```
$inhoud = strtolower(html_entity_decode
                                  ($inhoud));
$inhoud = preg_replace('/&#?\w+;/', ' ',
                                  $inhoud);
if (!get_magic_quotes_runtime())
            $inhoud = addslashes($inhoud);
$inhoud = preg_replace('/\s+/', ' ', $inhoud);
```

De eerste twee regels zijn bedoeld voor specifieke HTML-constructies zoals @euml; of ë. Het commando html_entity_decode() zorgt voor het converteren naar speciale tekens als ë. Vervolgens ruimt de

reguliere expressie op wat de ingebouwde functie laat staan. Tussendoor zorgt strtolower() ervoor dat alle letters klein worden, omdat er bij het zoeken geen onderscheid wordt gemaakt tussen hoofd- en kleine letters. Ook worden aanhalingstekens met addslashes() ongevaarlijk gemaakt. In dit geval geldt de PHP-instelling magic_quotes_runtime, omdat de $inhoud niet via get, post of cookie is binnengekomen. De reguliere expressie \s+ zoekt naar white spaces, oftewel spaties en regelafbrekingen, en vervangt die door een eenvoudige spatie. Dat maakt het zoeken een stuk makkelijker, want meestal worden samengestelde woorden op de woordgrens afgebroken.

Om bij de zoekresultaten iets bruikbaars weer te geven, destilleert de volgende regel de inhoud van de <title>-tag:

```
$titel = preg_match('#<title>(.+?)</
    title>#', $inhoud, $hits)? $hits[1] : $pad;
```

De reguliere expressie tussen de #-tekens achterhaalt de titel van de pagina. Daarbij is .+? een bijzondere *wildcard*: het staat voor een willekeurig aantal (+), maar liefst zo weinig mogelijk (?) willekeurige (.) tekens. Het zoeken stopt bij de volgende tag </title>.

De ronde haken vangen de gevonden titel af en zetten die in de array $hits. In deze array is met name element nummer 1 interessant – het eerste element $hits[0] bevat immers de <title>-tag. Als er geen titel is, omdat het bijvoorbeeld een tekstdocument of een minder goed geconstrueerde HTML-pagina is, dan gebruiken we het pad in plaats van de titel.

Het indexeerscript is alleen geïnteresseerd in woorden die zich buiten HTML-tags bevinden en in woorden die tussen <script>- en <style>-tags staan. Andere tags kun je in één keer lozen, maar voor deze gevallen is wat handwerk nodig:

```
$verwijderen = array('script', 'style');
foreach ($verwijderen as $tag) $inhoud =
  preg_replace('#<' . $tag . '\b.+?</' . $tag .
                                '>#', '', $inhoud);
```

Het zoekpatroon dat het script gebruikt om alle tags na te lopen, lijkt op het bovenstaande. Alleen \b is nieuw. Dit zorgt ervoor dat de tagnaam hier eindigt, omdat er bijvoorbeeld een spatie of een vishaak volgt. Anders zou het zoeken naar <b>… ook <body> opleveren. Deze expressie gaat de mist in als de tags genest zijn, zoals dat bij en <ul> mogelijk is, maar bij <script> en <style> zal dat niet het geval zijn.

Bonuspunten per zoekterm

Als volgende moeten we gaan nadenken over het wegen van de zoekresultaten. Het eenvoudigste zou zijn om gewoonweg het aantal keer dat een woord voorkomt te tellen. Bovendien moet tekst tussen bepaalde tags meer punten krijgen, bijvoorbeeld als het een header of een link betreft:

```
$bonus = array('title' => 6, 'h1' => 6, 'h2'
=> 5, 'h3' => 4, 'h4' => 3, 'h5' => 2, 'h6'
=> 1, 'a' => 1, 'b' => 1);
```

Een <h3>-header levert dus 4 punten op. Als dat ook nog een link is, komt daar een extra punt bij. Het script loopt alle tags één voor één door en indexeert de bruikbare tekst die tussen de tags staat. Vervolgens wordt de hele tekst geanalyseerd en levert ieder voorkomend woord een punt op.

```
foreach (array_keys($bonus) as $tag) {
  preg_match_all('#<' . $tag . '\b.+?</' .
                $tag . '>#', $inhoud, $hits);
  foreach ($hits[0] as $el)
          woordwaarde($el, $bonus[$tag]);
}
woordwaarde($inhoud, 1);
```

De buitenste foreach-loop leest de sleutels van de array $bonus in, oftewel de elementnamen. De volgende regel laat op de $inhoud dezelfde reguliere expressie los als boven. Ook hier geldt het nadeel dat geneste of niet afgesloten tags problemen zullen opleveren. In dit geval gebruiken we echter de preg_match_all()-functie, die alle vindplaatsen in de geneste array $hits() zet. Specifieker: in de array $hits[0]. De tweede foreach-loop roept voor iedere waarde in die array een functie aan die we nog moeten maken, met daarbij de bonuswaarde als tweede argument.

Ten slotte roepen we de functie woordwaarde() nog een keer aan voor de hele tekst van het document. Alle woorden krijgen hiermee een punt voor elke keer dat ze op de pagina voorkomen.

Bereken de 'PageRank'

De functie woordwaarde() moet de tekststring eerst ontdoen van alle HTML-code en tekens die niets met woorden te maken hebben. Vervolgens moeten betekenisloze stopwoorden er uitgehaald worden en moeten de geëxtraheerde woorden naar de database worden geschreven. Om de meegegeven $_tekst daarop voor te bereiden, heb je slechts drie regels nodig:

```
function woordwaarde($_tekst, $_score) {
  $_tekst = strip_tags($_tekst);
  $_tekst = preg_replace('/\W+/', '
                                ', $_tekst);
  $woorden = preg_split('/\s+/',
                          trim($_tekst));
  ...
}
```

Het commando strip_tags() verwijdert niet alleen de HTML-declaraties, maar ook eventuele PHP-code. De reguliere expressie \W geldt voor alle bijzondere tekens, en dan met name voor leestekens. Met preg_split() delen we de $tekst op bij de spaties (\s+) en schrijven we het resultaat in de variabele $woorden. Van tevoren hebben we met trim() eventuele spaties aan het begin en het einde van de string verwijderd.

Voordat we de woorden in de database opslaan, moeten we eerst controleren of ze zinvol zijn. Het script zal bijvoorbeeld de afkorting 'b.v.' als twee aparte letters beschouwen. In een zoekindex is dat vooral onhandig, dus gooien we woorden van 1 letter weg:

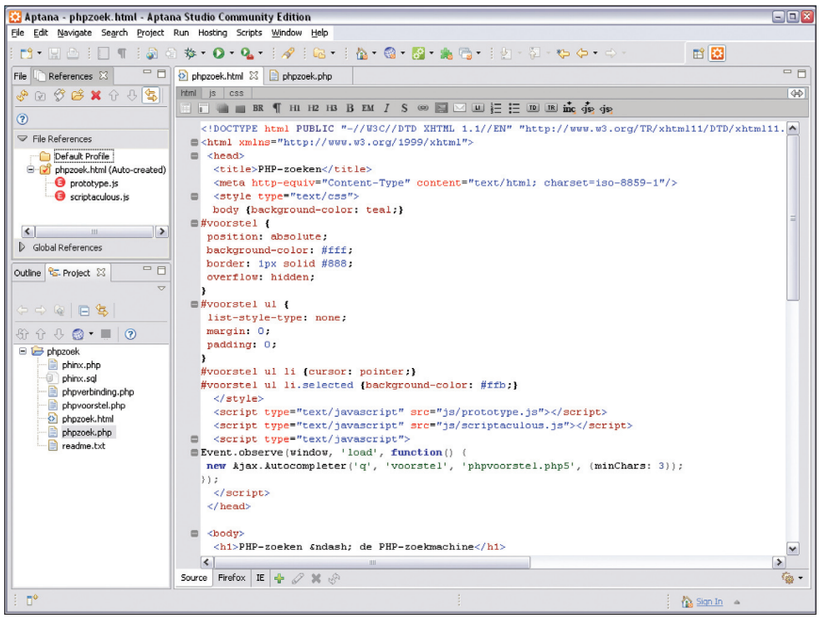
```
foreach ($woorden as $woord) {
  if (strlen($woord) < 2) continue;
  ...
}
```

De functie strlen() levert de woordlengte. Als die kleiner is dan 2, wordt de uitvoer van de betreffende loop door continue afgebroken en gaat de loop verder met het volgende woord. Ook getallen willen we niet indexeren:

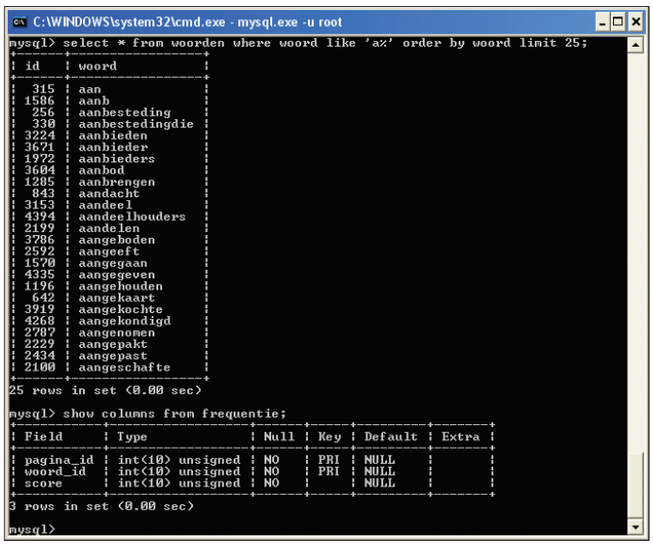
```
if (is_numeric($woord)) continue;
```

Stopwoorden

En dan hebben we nog de lijst met stopwoorden, ofwel de woorden die zoektechnisch niets betekenen. Die kun je het beste redelijk vooraan in het script zetten, zodat je hem makkelijk kunt bijwerken.



Het is even wennen om te werken met de gratis ontwikkelomgeving Aptana Studio, maar daar weegt de aanwezigheid van syntax-highlighting, code-completion en een debugger ruim tegenop.



Via de Opdrachtprompt kun je de MySQL-console starten. Dit is de meest directe manier om met de database te communiceren.

Je zou de lijst in een later stadium ook in de database kunnen zetten. Maar nu even niet:

```
$stopwoorden = array('de', 'het', 'een', 'van',
'en', 'met', 'of', 'als');
```

De geldigheid van variabelen is in PHP-functies standaard afgeschermd. Functies hebben geen toegang tot de variabelen van de code die hen aanroept en omgekeerd. Maar omdat we niet bij elke functieaanroep de hele array als argument willen hoeven meegeven, is het eleganter om de variabele als global te declareren:

```
function woordwaarde($_tekst, $_score) {
  global $stopwoorden;
  ...
}
```

PHP heeft een wat zonderlinge logica in de omgang met variabelen. De functie zorgt er zelf voor dat hij volledige toegang heeft tot variabelen van de aanroepende code. De uiteindelijke controle wordt gedaan in een foreach-loop met behulp van de functie in_array();

```
if (in_array($woord, $stopwoorden)) continue;
```

Het opslaan van de woorden in de database of het ophalen van de id van al opgeslagen woorden, gebeurt op dezelfde wijze als boven bij de tabel pagina:

```
foreach ($woorden as $woord) {
  ...
  $resultaat = mysql_query('SELECT id FROM
  woorden WHERE woord = "' . $woord . '"');
  $record = mysql_fetch_object($resultaat);
  if (!($woord_id = $record->id)) {
    mysql_query('INSERT INTO woorden
          (woord) VALUES ("' . $woord . '")');
    $woord_id = mysql_insert_id();
  }
  ...
}
```

Kruiskoppeling

Dan resteert nog het koppelen van de woorden aan de pagina's, waarbij we ook het gewicht moeten bewaren. Het zou omslachtig zijn als we die gegevens bij iedere aanroep van woordwaarde() en elke nieuwe waarde voor $_score in de database moesten gaan bewaren. Je kunt die gegevens beter bufferen in een array:

```
$woordfrequentie[$woord_id] += $_score;
```

Deze array moet echter wel blijven bestaan als de functie voltooid is, dus maken we ook hier een global variabele van. Dat kun je het beste meteen aan het begin van de functie doen. Dat deze variabele in de aanroepende code nog helemaal niet bestaat, is geen probleem – PHP initialiseert hem automatisch.

Voor de eerste alfaversie van het indexeerscript ontbreekt nu alleen nog wat code om de inhoud van $woordfrequentie in de database te zetten. Dat doe je nadat woordwaarde() voor de laatste keer is doorlopen.

```
...
woordwaarde($inhoud, 1);
mysql_query('DELETE FROM frequentie
        WHERE pagina_id = ' . $pagina_id);
foreach ($woordfrequentie as $woord_id =>
                                $score) {
  mysql_query('INSERT INTO frequentie
      (woord_id, pagina_id, score) VALUES
(' . $woord_id . ', ' . $pagina_id . ', ' . $score
                                . ')');
}
```

Aangezien we de database niet iedere keer weer updaten, gooien we de oude resultaten eerst gewoon weg met een DELETE-query voor alle frequentie-records van de betreffende pagina. Met een foreach-loop doorlopen we $woordfrequentie en voegen we voor ieder record een INSERT-query uit.

Ombouw

We zitten nu op zo'n 50 regels en 2.5 kB code, en voor een fatsoenlijk indexeerscript blijkt dat wel voldoende. Al ontbreken er nog wat dingen die je van zo'n script zou verwachten. Het is bijvoorbeeld niet te doen om elke pagina apart door het script heen te jagen. Gelukkig zijn computers voor zulk dom werk uitgevonden.

Om dit probleem op te lossen, laten we de mappenstructuur van de website doorploegen. Vervolgens roepen we voor ieder bestand met de juiste extensie het tot nu geschreven script op. Om dat te doen moeten we het feitelijke indexeeralgoritme in een functie stoppen. Het script blijft daarmee werken als voorheen, tenzij het de parameter pad=alles krijgt meegegeven.

Ook het inleesscript zit in een functie. Dat is nodig omdat het zichzelf moet kunnen oproepen bij het doorzoeken van submappen. Daarbij vindt de eerst aanroep plaats op het root-niveau van de website:

```
if ($pad == 'alles') {
  map_inlezen('.');
} else {
  indexeren($pad);
}
```

De indexeerfunctie krijgt het pad als argument mee en gebruikt een paar globale variabelen. Na het inlezen van het bestand gaat de code verder met wat we al eerder hadden:

```
function indexeren($pad) {
  global $verwijderen, $bonus;
  $inhoud = @ file_get_contents($pad);
...
}
```

De functie map_inlezen() berust voor een groot deel op de PHP-functies voor het inlezen van mappen: opendir() en readdir(). De eerste maakt een map-handle die de tweede gebruikt om alle bestanden na elkaar in te lezen:

```
function map_inlezen($_map) {
  $mh = opendir($_map);
  while (($bestand = readdir($mh)) !==
                                false) {
    ...
  }
}
```

Ook hier zorgt !== ervoor dat de lus niet wordt afgebroken bij een bestand met de naam '0'.

Mappen crawlen

Een map bevat niet alleen bestanden, maar ook de pseudomappen '.' en '..'. Dit soort mappen kun je het beste uit je index weren met een reguliere expressie die in de while-loop controleert of de bestandsnaam met een punt begint. Daarmee houd je bovendien ook meteen allerlei verborgen configuratiebestanden zoals .htpasswd buiten je index.

```
if (preg_match('/^\./', $bestand)) continue;
```

Als $bestand in werkelijkheid dan een map is – wat je met is_dir() kunt testen – dan doe je een nieuwe aanroep van map_inlezen() met het complete pad. Zo niet, dan ga je verder met de functie indexeren().

```
$pad = $_map . '/' . $bestand;
if (is_dir($pad)) {
  map_inlezen($pad);
} else {
  echo "<p>Indexeer $pad</p>\n";
  indexeren($pad);
}
```

Vervolgens worstelt deze crawler zich op magische wijze door de hele mapstructuur heen en wordt de database gevuld. Je krijgt nu alleen wel verkeerde scores, omdat de waarden van $woordfrequentie na het doorlopen van een pagina blijven bestaan en mee worden opgeteld. Door aan het begin van de indexeren()-functie met twee toevoegingen de array te resetten, lossen we dat probleem op.

```
global ..., $woordfrequentie;
$woordfrequentie = array();
```

Indexeerbaar of niet?

Een andere schoonheidsfout van de crawler is dat die iets te enthousiast te werk gaat. Het is niet erg zinvol om stylesheets,

scripts en zelfs hele afbeeldingen te gaan indexeren. Om die reden is het essentieel een lijst van bestandsnamen of extensies te hebben die je expliciet wel of niet wilt indexeren. Zo'n lijst kan er bijvoorbeeld zo uitzien:

```
$includeren = array('.html', '.htm', '.php',
                                    '.txt', '.xml');
$uitsluiten = array('afbeeldingen',
                                    'phinx.php');
```

De uitsluiten-lijst kan ook namen van mappen bevatten. Dat is erg belangrijk, aangezien onze crawler zich niets aantrekt van de toestemmingen die door de .htaccess worden geregeld. Uit esthetische overwegingen moeten deze lijsten weer aan het begin van het script staan en als global zijn gedefinieerd, zodat we ze in de functie map_inlezen() kunnen gebruiken.

Deze inclusie- en uitsluitcontrole zet je in een while-loop, en wel voor de regel if (is_dir($pad)) . . . . Twee foreach-loops lopen de lijsten door en vergelijken het complete $pad met de lijstitems. Hier de code voor de uitsluitcontrole:

```
foreach ($uitsluiten as $bestandsnaam) {
  if (preg_match('/' . $bestandsnaam . '$/',
                                    $pad)) continue 2;
}
```

De preg_match()-functie vergelijkt de lijstitems met $pad en de reguliere expressie test met de $-marker of de zoekterm aan het eind van de string staat. De lijst kan dus extensies of complete paden bevatten. Door continue 2 gaan we verder met de loop die om de foreach-loop staat, in dit geval while (readdir($mh)). Het script springt dus meteen naar het volgende bestand.

Voor de inclusielijst markeer je met de variabele $indexeerbaar ieder bestand in eerste instantie als verdacht.

```
$indexeerbaar = false;
foreach ($includeren as $extensie) {
  if (preg_match('/' . $extensie . '$/', $pad)) {
    $indexeerbaar = true; break;
  }
}
```

Het break-commando breekt de succesvol doorlopen testlus af. Na een andere kleine aanpassing houdt het script ook rekening met de $includeren-lijst:

```
if (is_dir($pad)) {
  map_inlezen($pad);
} else if ($indexeerbaar) {
  indexeren($pad);
}
```

Indexeren ondanks limieten

Als je deze crawler op een website met een paar grotere tekstbestanden loslaat, dan loop je mogelijkerwijs tegen een lastig probleem aan. De PHP-interpreter houdt er namelijk na een bepaalde (meestal door de provider ingestelde) tijd mee op en eindigt dan met een waarschuwing in de trant van 'Fatal error: Maximum execution time of 60 seconds exceeded in phinx.php on line 65'. En dan blijven er pagina's over die niet geïndexeerd zijn.

Het script moet dus een mogelijkheid hebben om alleen die pagina's te indexeren die nog niet aan de beurt zijn geweest. Omdat een pagina wellicht niet helemaal, maar slechts ten dele geïndexeerd kan zijn, moet het script het begin en einde van iedere schrijfactie ook als zodanig kenmerken. Daar heb je ten eerste een nieuw databaseveld voor nodig:

```
ALTER TABLE paginas ADD COLUMN compleet
BOOL NOT NULL DEFAULT 0;
```

De nieuwe modus 'alleen_nieuw' is bijna identiek aan 'alles'. De variabele $alleen_nieuw komt in de functie indexeren() als global te staan en wordt gebruikt om die twee uit elkaar te kunnen houden:

```
if ($pad == 'alleen_nieuw') {
  $pad = 'alles';
  $alleen_nieuw = true;
}
if ($pad == 'alles') ...
```

De SQL-query in de indexeren()-functie kijkt voor het ophalen van de pagina_id ook naar het veld compleet:

```
$resultaat = mysql_query('SELECT id,
compleet FROM paginas WHERE url = "' .
$pad . '"');
```

Als deze query geen hits oplevert, verandert er verder niets. MySQL zet bij het aanmaken van een nieuw record de standaardwaarde null in de kolom compleet. Als de pagina al bestaat, moet het script het indexeren afbreken als het in de $alleen_nieuw-modus loopt. Als later een al bestaande pagina geïndexeerd moet worden, zet het script met behulp van een UPDATE-query de waarde van compleet op null.

```
if ($pagina_id = $record->id) {
  if ($alleen_nieuw && $record->compleet)
                                    return;
  mysql_query('UPDATE paginas SET
    compleet = 0 WHERE id= ' . $pagina_id);
} else {
  mysql_query('INSERT ...
  ...
}
```

Bij de eerste if-voorwaarde gaat het niet om een vergelijking tussen $pagina_id en $record_id, want dan zou er == gestaan hebben in plaats van =. Het gaat om een toewijzing: er is aan de voorwaarde voldaan als $record_id een waarde heeft. Nu moeten we alleen aan het einde van de functie de waarde van compleet weer op 1 zetten:

```
function indexeren($pad) {
  ...
  mysql_query('UPDATE paginas SET
compleet = 1 WHERE id= ' . $pagina_id);
}
```

Opruimen

Als je het script met de parameter pad=alleen_nieuw oproept, kun je ook een website met een groter aantal pagina's in de database-index krijgen. Er is echter nog een laatste probleem voordat het script echt bruikbaar is. Er bestaat namelijk geen mogelijkheid om oude gegevens uit de database te verwijderen. Pagina's die allang niet meer bestaan of restanten van eerdere code-experimenten blijven tot in lengte van dagen in de database staan. Er worden immers alleen items verwijderd in $woordfrequentie. Met een paar extra regels code laat je het script in de modus alles de boel lekker opruimen. Dit werkt niet bij alleen_nieuw.

```
if ($pad == 'alles') {
  if (!$alleen_nieuw) {
    mysql_query('TRUNCATE paginas');
    mysql_query('TRUNCATE woorden');
    mysql_query('TRUNCATE
                        woordfrequentie');
  }
  map_inlezen('.');
}
```

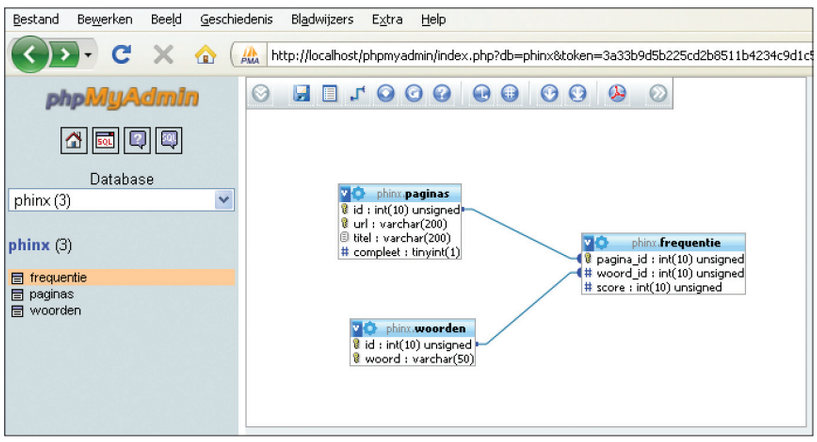
De TRUNCATE-commando's resetten de tabellen naar een maagdelijke toestand. Vervolgens begint het script van voren af aan met het vullen van de database.

Het zoekformulier

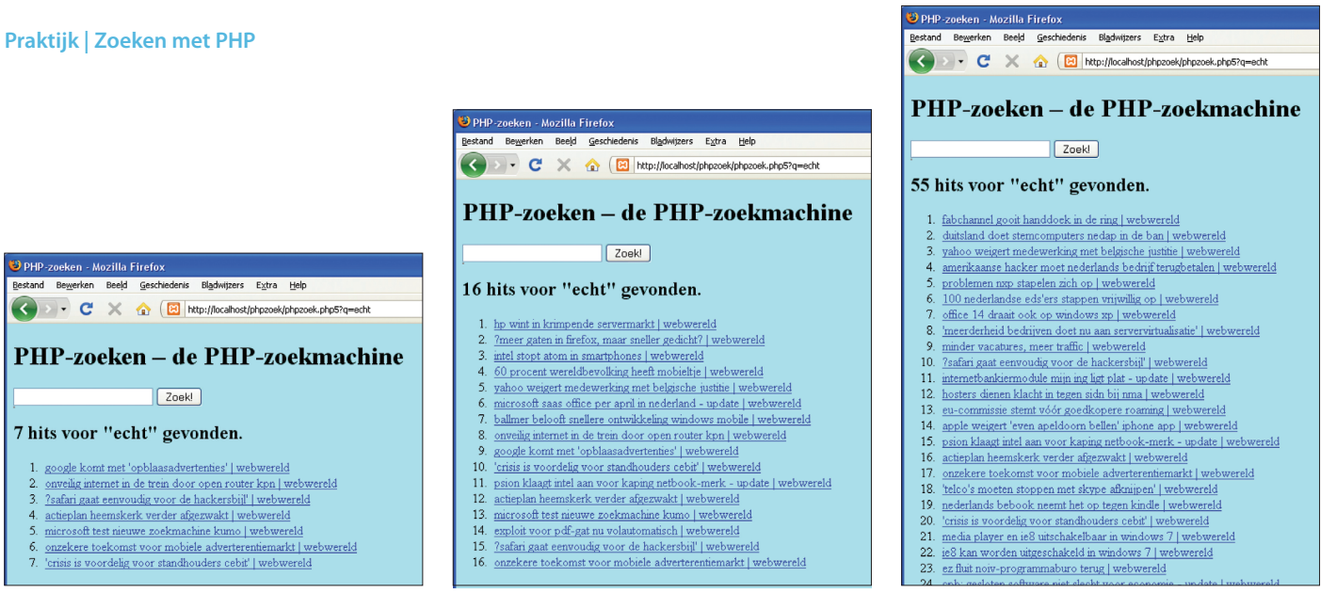
Met dit alles kun je nu een bruikbare index genereren. Nu heb je nog een script nodig om deze index te doorzoeken. Dit indexeerscript moet alleen niet voor iedereen toegankelijk zijn, en het script phpzoek.php juist weer wel. Daarbij moet je dus in de gaten houden dat je systeem zwaarder belast wordt en je mogelijk issues krijgt met de beveiliging.

Het principe is simpel: je hebt een invoerveld nodig, een SQL-query en een lijst resultaten. Het invoerveld kan er bijvoorbeeld zo uitzien:

```
<form action="phpzoek.php"
                        method="get">
  <p>
```



Nieuwere versies van phpMyAdmin kunnen zelfs de relaties tussen tabellen grafisch weergeven, naar eenvoudig voorbeeld van de MySQL Workbench en het professionele Rational Rose.



Vooral bij een kort zoekwoord kan het veel uitmaken of de zoekmachine exact moet zoeken (links), een wildcard aan het einde hangt (midden) of aan beide kanten (rechts).

```
    <input type="text" size="25"
                    name="q"/><br />
    <input type="submit"
                    value="Zoek!"/>
  </p>
</form>
```

De PHP-code die door dit formulier wordt aangestuurd, leest eerst het zoekwoord uit en maakt dan verbinding met de database:

```
if ($zoekstring = trim($_GET['q'])) {
  if (!get_magic_quotes_gpc())
    $zoekstring = addslashes($zoekstring);
  require('db_verbinding.php');
  ...
}
```

De trim()-functie verwijdert eerst eventuele spaties aan het begin en het einde van de zoekstring. Voordat we verder gaan, controleren we de lengte van de zoekstring. Wellicht een overbodige voorzorgsmaatregel, maar afhankelijk van de serverconfiguratie kan het script wel tot een paar duizend tekens in de maag gesplitst krijgen.

```
if (strlen($zoekstring) > 50)
            die('Zoekopdracht te groot');
```

Meerdere zoektermen

De volgende taak bestaat uit het opdelen van de zoekwoorden als de gebruiker er meerdere heeft opgegeven:

```
$zwd = preg_split('/\s+/', $zoekstring);
```

Ook preg_split() kwam al eerder bij het indexeerscript aan bod. Dit had sneller gekund met explode(' ', $zoekstring), maar dat zou tot problemen leiden als de gebruiker meerdere spaties tussen de losse zoekwoorden heeft staan. Vervolgens gaat een lus alle items van $zwd af en haalt daar alle overbodige, merkwaardige of zelfs gevaarlijke tekens uit:

```
foreach ($zwd as $zw) {
  $zw = preg_replace('/\W/', '', $zw);
  if (strlen($zw) > 1)
            $zoekwoorden[] = $zw;
}
```

Het resultaat wordt in de nieuwe array $zoekwoorden gezet – als de zoekwoorden tenminste uit 2 of meer tekens bestaan. Als blijkt dat er geen zoekwoorden overblijven, kunnen we de zoekactie beëindigen.

```
if (!$zoekwoorden) {
  echo 'Geen zoekwoorden opgegeven!';
  exit;
}
```

Ook het tegendeel is niet wenselijk: als er een stuk of tien zoekwoorden overblijven, krijgt de database het wel erg druk. De functie array_splice() reduceert het aantal zoekwoorden tot 4:

```
if (count($zoekwoorden) > 4)
        array_splice($zoekwoorden, 4);
```

De zoekquery

Je bent nu bij het punt dat de zoekopdracht geformuleerd is en de database doorzocht kan worden. Hierbij zijn nog een aantal keuzes te maken. Moet de database een hit melden als één van de zoektermen gevonden is, of moeten alle zoekwoorden in de tekst voorkomen? Wat moet het resultaat zijn als de zoekterm 'dag' is? Ook 'maandag' en 'dageraad'? En last but not least: wat te doen als de gebruiker 'reimwoord' intypt?

Het makkelijkst is om met het eenvoudigste geval te beginnen: een enkele zoekterm en een simpele stringvergelijking. De SQL-code ziet er dan ongeveer zo uit:

```
$sql = 'SELECT p.url, p.titel FROM paginas
p, woorden w, frequentie f WHERE f.woord
= "' . $zoekwoord[0] . '" AND f.woord_id
= w.id AND f.pagina_id = p.id ORDER BY
f.score DESC';
```

Bij query's over meerdere tabellen is het aan te raden afkortingen zoals paginas p (kort voor: paginas AS p) te gebruiken. De WHERE-clausule vergelijkt de woordenlijst met het eerste zoekwoord en maakt verbinding via de frequentie-tabel. De uitvoer wordt gesorteerd op een dalende (DESC) score in de frequentie.

```
$q = mysql_query($sql)
                or die(mysql_error());
```

Voordat je de resultaten laat zien, controleer je eerst met mysql_num_rows() of die er wel zijn:

```
if ($aantal = mysql_num_rows($q)) {
  echo '<h2>' . $aantal . ' hits voor &quot;'
. $zoekstring . '&quot; gevonden.</h2>'; //
resultaten ...
} else {
  echo '<h2>Geen hits voor voor &quot;' .
$zoekstring . '&quot;</h2>';
}
```

De gevonden resultaten zijn snel geschreven met mysql_fetch_object(), dat de hits regel voor regel inleest:

```
echo '<ol>';
while ($record = mysql_fetch_object($q))
  echo '<li><a href="' . $record->url . '">' .
            $record->titel . '</a></li>';
echo '</ol>';
```

En, of of?

Voor een eerste zelfgemaakte zoekmachine is je huidige code al heel keurig, maar vooral het werken met meerdere zoektermen ontbreekt nog. Een eerste variant is via een OF-functie. Bij de zoekstring 'een twee drie' komt de database met alle pagina's waar een van die drie woorden op staan. In SQL ziet dat er voor twee termen zo uit:

```
SELECT p.url, p.titel FROM paginas p,
woorden w, frequentie f WHERE (w.woord =
'een' OR w.woord = 'twee') AND f.woord_id
= w.id AND f.pagina_id = p.id GROUP BY
p.id ORDER BY f.score DESC;
```

De haakjes rondom de zoektermen zijn belangrijk, omdat er door de volgorde van AND en OR anders een onzinnig resultaat uit zou komen. Het groeperen via GROUP BY zorgt ervoor dat de hits niet vaker dan één keer voorkomen.

De bovenstaande zoekquery levert de juiste hits op, maar heeft een nadeel: je krijgt de score van de laatste zoekterm in plaats van de som van alle scores van de verschillende zoektermen. Met de standaardmethoden van SQL is dat blijkbaar niet voor elkaar te krijgen. Een mooie, maar bewerkelijke oplossing zou zijn om iedere zoekterm afzonderlijk te laten doorlopen en met PHP de resultaten bij elkaar te laten voegen. Dat zou met de functies array_merge() en array_unique() kunnen. Ook zou je als compromis met array_reverse($zoekwoorden) eerst de volgorde van de zoektermen kunnen omdraaien voordat je ze aan MySQL geeft. De belangrijkste zoektermen staan vermoedelijk namelijk vooraan.

Maar waarschijnlijk is het samenvoegen van de zoektermen

met een logische AND zinvoller. Dan verschijnt een pagina alleen als hit als het alle zoektermen bevat. Het voor de hand liggende idee om dan de OR in de laatste SQL-query door een AND te vervangen is te kort door de bocht: de database levert dan geen hits terug, omdat een woord niet tegelijkertijd een ander woord kan zijn.

Ook hier zou het mogelijk moeten zijn iedere zoekterm apart te doorlopen en dan met PHP samen te voegen met array_intersect(). Maar vanaf MySQL 4.1 is dat probleem ook zonder omwegen op te lossen:

```
SELECT p.url, p.titel FROM paginas p,
woorden w, frequentie f, (SELECT p.id, f.score
FROM paginas p, woorden w, frequentie f
WHERE w.woord = 'een' AND f.woord _id
= w.id AND f.pagina_id = p.id) j1 WHERE
w.woord = 'twee' AND f.woord _id = w.id
AND f.pagina_id = p.id AND p.id = j1.id
ORDER BY f.score + j1.score DESC;
```

De truc bestaat uit de tussen haakjes staande subquery met de naam j1, die als een gewone tabel in de FROM-lijst wordt behandeld. Deze subquery haalt voor de hoofdquery de pagina-id's en de scores van het eerste woord op om te kunnen sorteren. Met het deel WHERE j1.id = p.id krijg je dan de dwarsdoorsnede tussen de hoofd- en de subquery, die gesorteerd wordt op de som van de scores.

Goochelen met zoektermen

Op deze manier kun je ook meer dan twee zoektermen koppelen. Dan moet je in PHP wel eerst met een paar regels code de SQL-query in elkaar bouwen:

```
$sql = 'SELECT p.url, p.titel FROM paginas
                p, woorden w, frequentie f';
$where = ' WHERE w.woord = "' . array_
  shift($zoekwoorden) . '" AND f.woord_id
          = w.id AND f.pagina_id = p.id';
$order = ' ORDER BY f.score';
// extra zoekwoorden
$sql .= $where . $order . ' DESC';
```

De query wordt in drie delen opgedeeld. Zo kun je via een loop extra query's tussenvoegen voor meerdere zoekwoorden. De functie array_shift() verwijdert het eerste element uit de array en levert dit terug. Als je meer dan één zoekterm hebt ingevoerd, werkt het script een lus af:

```
while ($zw = array_shift($zoekwoorden))
{
  $nr = count($zoekwoorden);
  $sql .= ', (SELECT p.id, f.score FROM
paginas p, woorden w, frequentie f
WHERE w.woord = "' . $zw . '" AND
f.woord_id = w.id AND f.pagina_id =
p.id) j' . $nr;
  $where .= ' AND p.id = j' . $nr . '.id';
  $order .= ' + j' . $nr . '.score';
}
```

De while-loop haalt het eerste element uit de array, zolang er een voorradig is. $nr is een eenduidig nummer dat nodig is voor het identificeren van de subquery's. De volgende drie regels hangen een subquery, een WHERE-clausule en de som van de scores aan de query.

Fuzzy search

Tot nu toe levert de zoekmachine alleen hits op als een van de zoektermen ook precies zo in de tekst voorkomt. Dat is in de praktijk niet altijd even handig. Een eenvoudige uitbreiding is dan ook het toestaan van wildcards bij de zoektermen. In SQL kan dat door in de query's de operator = te vervangen door LIKE, waarbij de underscore _ voor één en het procentteken % voor een willekeurig aantal tekens staan.

```
... WHERE w.woord = "zoekwoord" AND ...
```

wordt dan bijvoorbeeld

```
... WHERE w.woord LIKE "zoekwoord%"
AND ...
```

De zoekmachine vindt bij 'kat' dan ook 'kattenbrokken'. Jammer alleen dat ook 'kater' een legitieme hit is. Een extra procentteken aan het begin van het zoekwoord maakt het aantal hits nog groter, omdat dan ook het woord 'skaten' matcht.

Daarnaast komen de zoekresultaten helaas meer dan eens in de lijst voor, namelijk voor ieder woord dat met de wildcards mogelijk is. Dit kun je ondervangen door achter SELECT het woord DISTINCT toe te voegen, zodat iedere hit maar één keer wordt vermeld. Alleen werkt het scoren dan niet goed meer, omdat hiermee alleen de waarde van één van de zoekvariaties wordt berekend.

Dit kun je alleen oplossen door het script voor iedere gevonden string een aparte query te laten starten, waarvan de scores dan via PHP opgeteld worden. Maar dat is wel een heel hoge prijs voor wat extra nauwkeurigheid. Dat geldt des te meer voor andere manieren om strings te vergelijken met PHP of in MySQL (zie kader).

Zoeken met suggesties

Het zoeken levert nu een flink aantal enigszins bruikbare hits op. Er zijn aardig wat mogelijkheden om het script verder uit te werken. Zo kun je proberen om wat meer Booleaanse logica in de zoekopdracht te verwerken (+, -, AND, OR, haakjes). Als je volledige teksten in de indexdatabase stopt, kun je ook op expliciete woordvolgordes zoeken door er bijvoorbeeld aanhalingstekens omheen te zetten. Ook kun je zo de exacte locatie van de gevonden zoekwoorden in de tekst aangeven door ze te markeren.

Ook aan het indexeren valt nog wat bij te vijlen. Zo kun je de inhoud van metatags analyseren of de alt-attributen van afbeeldingen. Ook kun je een groter gewicht toekennen aan woorden die eerder in de tekst voorkomen, of kun je rekening houden met de lengte van de tekst. In onze zoekmachine hebben we een voorkeur voor langere teksten, aangezien je dan eerder een volwaardige index hebt opgebouwd en meer zoekresultaten. Ook kun je meerdere hits opdelen in een aantal resultaatpagina's, zoals zoekmachines als Google dat doen.

Eén detail willen we nog implementeren, waar je de gebruiker aardig mee kunt imponeren. We gaan met behulp van Ajax een lijst toevoegen om het intypen van een zoekterm automatisch te laten aanvullen. Google had die functie jaren terug al onder de naam 'Google Suggest', en het is naderhand op heel wat plekken overgenomen. Sinds medio 2008

Slim tekstvergelijken

Zowel PHP als MySQL beschikken over geavanceerde technieken om teksten met elkaar te vergelijken. MySQL kan bijvoorbeeld niet alleen overweg met simpele wildcards (met LIKE), maar ook met reguliere expressies (REGEXP).

Het Soundex-algoritme (SOUNDEX) probeert zelfs woorden te vergelijken op grond van hun fonetische uitspraak, door klinkers en stemloze medeklinkers te negeren. Wat overblijft is de eerste letter van het woord, gevolgd door cijfers die de resterende medeklinkers moeten voorstellen. De 18 medeklinkers worden daarbij wel op zes hopen gegooid: de b, f, p en v zouden bijvoorbeeld allemaal een 1 worden. Het woord 'skater' valt op die manier uiteen in s, k, t, r en krijgt aldus code S236. Dezelfde code geldt echter voor 'schitteren' en zelfs voor 'squadron'. Dit komt doordat deze methode toegespitst is op het Engels, maar voor het Nederlands werkt het redelijk.

De MySQL-functie MATCH() … AGAINST() is geschikt voor het doorzoeken van grotere tekstbestanden. Voorwaarde is wel dat er in de database een FULLTEXT-index is aangemaakt (met FULLTEXT(kolom)). Daarmee kan de database op dezelfde manier de tekstbestanden afstruinen als ons script in dit artikel. Een query als SELECT MATCH(kolom) AGAINST('woord') FROM tabel levert dan een getal op dat een relatief gewicht voorstelt.

PHP kan ook een Soundex-algoritme toepassen, al is dit daar (net als de reguliere expressies) iets anders geïmplementeerd. Dan werkt metaphone() iets beter dan Soundex, al werkt die ook voornamelijk in het Engels.

Een eenvoudiger algoritme om twee strings op hun gelijkheid te beoordelen, is de Levenshtein-afstand (een algemene variant op de bekendere Hamming-afstand). Hierbij tel je iedere toevoeging, wijziging of verwijdering van een letter, die je nodig hebt om de ene string in de andere te wijzigen, als één stap. Het PHP-commando levenshtein('dropje', 'dropjes') levert de waarde 1 op, terwijl de afstand tussen 'Windows' en 'Linux' 5 is. Die begrippen hebben duidelijk minder met elkaar gemeen. De functie similar_text() bewandelt de omgekeerde weg. Hierbij krijgen woorden met een grotere overeenkomst in letters een hogere waarde.

Zoekmachines als Sphider, Sphinx en MnogoSearch kun je bovendien uitbreiden met de *stemming*-bibliotheek libstemmer, waarmee je – afhankelijk van de taal die je instelt – bijvoorbeeld ook teksten met het woord 'bomen' vindt als je op het woord 'boom' zoekt.

zit het ook standaard in de Engelstalige Google.

Het schrijven van zo'n Ajax-functie is nog niet zo eenvoudig. Gelukkig hebben anderen die moeite al gedaan, waaronder Thomas Fuchs, de auteur van het JavaScript-framework script.aculo.us. Om diens script te integreren, moet je het eerst downloaden (zie softlink). Kopieer prototype.js uit de lib-map en de inhoud van de src-map (eigenlijk zijn alleen scriptaculous.js, controls.js en effects.js nodig, de rest is optioneel) naar je webserver en integreer die in de HTML-header van je zoekpagina.

```
<script type="text/javascript"
          src="js/prototype.js"></script>
<script type="text/javascript"
        src="js/scriptaculous.js"></script>
```

De overige scriptaculous-bestanden worden vanzelf geladen. Verder hoef je maar weinig te veranderen aan de HTML-interface van de zoekpagina: het tekstinvoerveld heeft een unieke id nodig en in de buurt ervan moet een lege div-container met een eigen id staan:

```
<script type="text/javascript">
Event.observe(window, 'load', function() {
      new Ajax.Autocompleter('q', 'voorstel',
          'phpvoorstel.php', {minChars: 3});
});
</script>
```

Ajax.Autocompleter is een kant-en-klare scriptaculous-functie. Deze functie heeft vier argumenten nodig: de id's van het invoerveld en de container, de url van de pagina die de suggesties moet leveren en een optionele lijst van parameters. De minChar-parameter bepaalt dat het Ajax-script pas na de derde ingetypte letter voorstellen gaat zoeken. Deze Autocompleter-functie moet na het laden van de pagina beschikbaar zijn. Hiervoor zorgt de in Prototype-stijl geformuleerde regel Event.observe.

Testrun

Voordat we kunnen kijken hoe Autocompleter werkt, hebben we nog het bestand phpvoorstel.php nodig. Scriptaculous gaat er vanuit dat daar een -lijst in staat. De gebruikelijke HTML-overhead is niet nodig. Een paar regels HTML zijn voldoende om te testen:

```
<ul>
  <li>een</li>
  <li>twee</li>
</ul>
```

Bij het intypen van de derde letter in het invoerveld verschijnen dan beide items als voorstel. Hoe dat er allemaal uitziet, laat nog te wensen over. Scriptaculous zet de lijst wel op de goede plek, maar houdt er standaard geen rekening mee dat de achtergrond transparant is. Met de volgende CSS-instructies, die je met <style type="text/css">…</style> in de HTML-header van phpzoek.php zet of waar je met <link rel="stylesheet" href="…" /> naar verwijst, wordt het er in dit voorbeeld wat mooier op:

```
#voorstel {
  position: absolute;
  background-color: #fff;
  border: 1px solid #888;
  overflow: hidden;
}
#voorstel ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}
#voorstel ul li {cursor: pointer;}
#voorstel ul li.selected {background-color:
                                    #ffb;}
```

Deze stylesheet-instructies regelen niet alleen de achtergrond en de randen. Ook te lange woorden worden afgeknipt en opsommingtekens onzichtbaar gemaakt. De laatste twee regels veranderen de muispointer en geven het woord onder de muis een andere kleur.

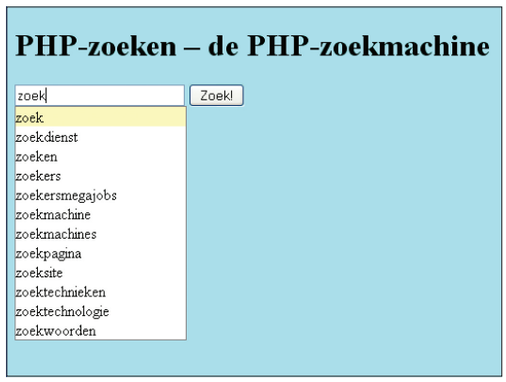
Omgaan met zoeksuggesties

In de praktijk bevat het bestand phpvoorstel.php natuurlijk niet alle mogelijke woorden, maar vist het ze uit de database. De eerste stap is het inlezen van de zoekterm, dat scriptaculous via de POST-methode doorgeeft:

```
<ul>
<?php
$zoek = trim($_POST['q']) or exit;
if (!get_magic_quotes_gpc())
                    $zoek = addslashes($zoek);
if (strlen($zoek) > 50) exit;
```

Ook hier treffen we dezelfde voorzorgsmaatregelen met addslashes() aan, omdat deze pagina rechtstreeks vanaf internet toegankelijk is. Stap twee is het opzetten van een databaseverbinding, maar dat is snel opgelost:

```
require('db_verbinding.php');
```



Het automatisch aanvullen van zoektermen is met een kant-en-klaar script zonder veel moeite te implementeren.

De databasequery zoekt naar woorden die beginnen met de inhoud van $zoek:

```
$q = mysql_query('SELECT woord FROM
woorden WHERE woord LIKE '" . utf8_
decode($zoek) . '%'");
```

Ook als de zoekpagina een andere tekenset gebruikt, komen de woorden toch in UTF-8 aan in het Ajax-script. Om de zoektermen te kunnen vergelijken met die in de database, moeten ze eerst naar ISO 8859-1 worden omgezet. Bij het weergeven van de hits moet het script de resultaten weer terug omzetten, omdat JavaScript anders een puinhoop maakt van trema's en accenten.

```
while($record = mysql_fetch_row($q))
  echo '<li>' . utf8_encode($record[0]) .
                                  "</li>\n";
?>
</ul>
```

Praktijk

Nu moet het script nog onder moeders vleugels vandaan, van de lokale ontwikkelomgeving naar de webserver. Deze code werkt zowel onder PHP4 als PHP5. Wat je in ieder geval moet wijzigen, zijn de instellingen van de databaseverbinding. In het ideale geval maak je gewoon twee verschillende scenario's in db_verbinding.php:

```
if ($_SERVER['HTTP_HOST'] == 'localhost') {
  mysql_connect('localhost','root','')
                    or die('Geen verbinding');
  mysql_select_db('phinx')
                     or die('Geen database');
} else {
  mysql_connect('jouw.provider.nl',
                  'inlognaam','wachtwoord')
                    or die('Geen verbinding');
  mysql_select_db('databasenaam')
                     or die('Geen database');
}
```

Hiermee werkt het script zowel op je lokale webserver als op internet en hoef je het niet steeds aan te passen.

Het kan voorkomen dat het indexeerscript alle trema's en accenten door een spatie vervangt. Dat komt door de instructie preg_replace('/\W+/', ' ', $inhoud) in de functie woordwaarde(). Als PHP door zijn configuratie namelijk niet weet dat het Nederlandse teksten betreft, gelden dit soort speciale karakters niet als letters. Dit kun je oplossen door aan het begin van het indexeer- en het zoekscript de volgende regel in te bouwen:

```
setlocale(LC_ALL, 'nl_NL');
```

Op die manier kun je de zoekmachine ook voor andere talen aanpassen.

Wie zoekt…

Onze zelfbouwzoekmachine heeft nog niet het comfort van een geïntegreerde Google- of Yahoo-zoekbalk en haalt het ook nog niet bij kant-en-klare zoekscripts. Je kunt er als webmaster dus nog alle kanten mee op, bijvoorbeeld qua weergave van de zoekresultaten.

Al met al heb je zo'n 150 regels PHP nodig om een zoekmachine te maken die een statische website volledig inleest, in losse woorden opdeelt en de geïndexeerde pagina's snel weer terugvindt. De mogelijkheid tot het automatisch aanvullen van de zoektermen met Ajax ziet er niet alleen gelikt uit, maar leidt ook tot sneller in beeld verschijnende resultaten. Die snelheid is in combinatie met een intuïtieve bediening sowieso de reden waarom zoekmachines bij surfers zo populair zijn. (nkr)

Softlink 0905122 c't