Ausgabe **5.08** August | September www.phpmagazin.de S&S

Mit CD!

PHPmagazin

Deutschland 9,80 € Österreich 10,80 € | Schweiz 19,20 SFr
Niederlande 11,25 € | Luxemburg 11,25 €

PostgreSQL
Eigene Suchmaschine mit Volltextsuche

PHP-Security
Intrusion Detection mit PHPIDS

Enterprise-Wiki
DokuWiki als Wissensdatenbank im Unternehmen

PHP ohne Code
Model-driven Development mit dem Zikula-Framework

State of the Art

DojoX – Trends und Komponenten für das Web 2.0

Drei. Punkt. Null.

Mit PHP-Keywords ein semantisches Netz aufbauen

INHALT

Video: Dynamic Languages World Europe 2008

Datenträger enthält nur Lehr- oder Infoprogramme

4 195649 609809 05

Volltextsuche mit tsearch2 in der PostgreSQL-Datenbank

Schnüffler

In unserem Artikel „Umstieg von MySQL auf PostgreSQL – die ersten Schritte“ in der letzten Ausgabe des PHP Magazins haben wir Ihnen ja bereits versprochen, das Thema PostgreSQL weiter zu vertiefen. Here we go! Wir zeigen Ihnen, wie Sie mit der PostgreSQL und der Erweiterung tsearch2 eine Suchmaschine über volltextindizierte Inhalte bauen.

▶ von Thomas Pfeiffer und Andreas Wenk

Eine Info gleich vorweg: tsearch2 ist seit der PostgreSQL-Version 8.3 in den Kernel integriert. Wir behandeln hier dennoch die Version 8.1 – aus zwei Gründen: Zum einen gehen wir davon aus, dass die meisten User noch nicht über die neueste Version der PostgreSQL verfügen, zum anderen arbeiten wir selbst mit Version 8.1 und denken, dass der „Lernwert“ höher ist, wenn Sie die Erweiterung und Einrichtung von der Pike auf lernen und umsetzen. Bei Version 8.2 können Sie tsearch2 aus dem *Contrib*-Paket, wie hier beschrieben, nutzen.

Wir können Sie zudem beruhigen, denn in der Onlinedokumentation von PostgreSQL 8.3.1 [1] ist nachzulesen, dass die Integration von tsearch2 in den Kernel der Datenbank rückwärtskompatibel ist. Das bedeutet, es gibt in PostgreSQL 8.3 Alias-Funktionen für die Funktion von tsearch2 vor der Integration in den Kernel der Datenbank. In einigen Fällen müssen Sie dann noch ein paar Anpassungen vornehmen – vor allem wenn Sie die tsearch2-internen Tabellen inhaltlich ändern, so wie wir es hier im Artikel beschreiben. Der Umfang ist aus unserer Sicht allerdings sehr überschaubar.

Wir behandeln im Folgenden alle notwendigen Schritte, um eine Datenbank mit der tsearch2 Engine zu erweitern und zu konfigurieren. Außerdem zeigen wir Ihnen, wie Sie mit dieser Engine Daten in die Datenbank schreiben und die Datensätze dann durchsuchen können. Zur Unterstützung und Veranschaulichung haben wir eine Beispielapplikation erstellt. Diese können Sie unter [2] herunterladen.

Die Nicht-Linux-User unter Ihnen mögen es uns nachsehen, dass wir als Grundlage ein Linux-Debian-System verwenden und daher keine Beschreibungen für andere Systeme (Windows) zur Verfügung stellen können.

Motivation und Grundprinzip

Spätestens, seitdem „googeln“ auch im Duden steht, ist die Suche nach Informa-

tionen im Internet eine der am häufigsten genutzten Aktionen. Dabei wird ein Suchergebnis erwartet, das der Anfrage sehr nahe kommt. Das bedeutet, die Übereinstimmung zwischen Suchanfrage und Suchergebnis soll inhaltlich so groß wie möglich sein, außerdem soll das Ergebnis nach Relevanz gewichtet sein und dargestellt werden. Natürlich ist auch eine hundertprozentige Übereinstimmung möglich.

Was muss mit Dokumenten geschehen, bevor Sie in der Datenbank abgelegt werden? Bevor sie durchsucht werden können, müssen ihre Inhalte aufbereitet werden. Zum einen werden „Stopwords" wie *ist*, *und*, *ein*, *mit* usw. entfernt, zum anderen wird *Word Stemming* vorgenommen. Dabei werden Worte auf ihren lexikalischen Stamm reduziert, d.h. von ähnlichen Worten wird deren Wortstamm gespeichert. Dies nur als Beispiel für die Aufbereitung von Dokumenten sein. Der aber wahrscheinlich wichtigste Aspekt in diesem Zusammenhang ist das Extrahieren von Keywords – also der Worte, die den Inhalt des Dokuments letztendlich beschreiben.

In der tsearch2 Engine kommt das so genannte „gewichtete Vektorraummodell" zum Einsatz. Dahinter steckt der Grundgedanke, die aufbereiteten Dokumentinhalte und die Suchanfrage in Vektoren umzuwandeln. Die Länge des Vektors beschreibt die Häufigkeit des Vorkommens der einzelnen Keywords im Dokument. Bei der Suchanfrage wird dann der Vektor der Suchanfrage mit den Vektoren der Dokumente verglichen. Je näher der Vektor der Suchanfrage am Vektor eines Dokuments liegt, desto höher ist dessen Relevanz und somit dessen Gewichtung.

Um einen tieferen Einstieg in die Materie zu erhalten, verweisen wir auf das Buch „Suchmaschinenoptimierung" von Sebastian Erlhofer und seine vorzügliche Einführung in die technischen Grundlagen von Suchmaschinen (Kap. 3 – 5).

Aufbau der tsearch2 Engine

Die beiden wichtigsten Funktionen in der tsearch2 Engine sind zum einen eine Funktion namens *to_tsvector* und zum anderen eine Funktion namens *to_tsquery*, außerdem gibt es einen eigenen Datentyp namens *tsvector*. Die tsearch2 Engine basiert im Prinzip auf vier Funktionalitäten:

1. Aus den Textdaten, die in eine Tabelle geschrieben werden sollen, werden alle einzelnen Worte als *tsvector* in einer separaten Spalte (z.B. *vector*) der Tabelle gespeichert.
2. Auf die Spalte, in der die Vektordaten gespeichert wurden, wird ein Index gesetzt.
3. Innerhalb der Suchabfrage wird durch eine *tsquery* die Vektorspalte durchsucht und die entsprechenden Ergebnisse bzw. Zeilen, auf die die Suche „matcht", werden zurückgegeben.
4. Das Ergebnis wird ausgegeben, wobei dies z.B. absteigend nach Treffergenauigkeit (*ranked*) geschehen kann (und wohl sollte).

Zusammengefasst bedeutet das, wir nutzen die Funktion *to_tsvector,* um die Daten in der Datenbank zu speichern, und die Funktion *to_tsquery,* um die Daten aus der Datenbank auszulesen. Wir werden in den folgenden Abschnitten genauer sehen, wie dies umgesetzt wird. Aber zuerst installieren wir die tsearch2 Extension.

Installation der tsearch2 Extension

Wir beschreiben im Folgenden die Installation der tsearch2 Extension auf einem Debian-Sarge- oder Etch-System. Dabei erklären wir nicht noch einmal die Installation einer PostgreSQL. Sie können das im oben erwähnten Artikel in der Ausgabe 04.08 des PHP Magazins nachlesen.

Die tesarch2 Extension befindet sich im *Contrib*-Paket der PostgreSQL von Debian. Die Installation erfolgt wie unter Debian gewohnt:

```
root: apt-get install postgresql-contrib-8.1
```

Das war's auch schon. Wichtig ist zu wissen, dass ein SQL Dump namens *tsearch2.sql* im Verzeichnis *root: /usr/share/postgresql/8.1/contrib/$* abgelegt wurde. Dazu gleich mehr.

Jetzt müssen wir noch ein paar Dinge von Hand erledigen. Erstellen wir uns zuerst eine Datenbank zum Spielen, die uns für den Rest des Artikels begleiten wird. Wir können dafür z.B. das Administrationstool pgAdmin [3] nutzen oder den Command-Line Client der PostgreSQL.

Wir nutzen hier den Client und sind als *root* angemeldet. Zuerst wechseln wir vom *root*-Benutzer zum Standard-Postgre SQL-Benutzer *postgres*:

```
root: su postgres
```

Im nächsten Schritt erstellen wir unsere Datenbank, wobei in der Befehlszeile *-O* für O*wner* (Eigentümer) steht und vom Datenbanknamen gefolgt wird:

```
postgres: createdb -O postgres tsearch
CREATE DATABASE (Ausgabe)
```

Dann schauen wir, ob alles geklappt hat, und prüfen die Kodierung. Dazu rufen wir mit dem Dienstprogramm *psql* die Datenbank *tsearch* auf:

```
postgres: psql tsearch
tsearch=#
tsearch=# \l
 Liste der Datenbanken
  Name   | Eigentümer | Kodierung
-----------+------------+-----------
tsearch   | postgres   | UTF8
```

Das sieht gut aus. Übrigens können sie innerhalb des Command Clients mit *\h* die Hilfe für SQL-Anweisungen aufrufen und mit *\?* die Hilfe für interne Befehle einsehen. Mit *\q* beenden Sie den Client, und das tun Sie jetzt auch bitte. Falls Sie sich noch nicht in dem Verzeichnis mit dem SQL Dump *tsearch2.sql* befinden, wechseln Sie jetzt mit folgendem Befehl dorthin:

```
postgres: cd /usr/share/postgresql/8.1/contrib/
```

Denn jetzt spielen wir den Dump *tsearch2.sql* in unsere gerade erstellte Datenbank *tsearch* ein. Aber einen Check nehmen wir noch vor:

```
postgres:/usr/lib/postgresql/8.1/bin/pg_controldata
    /var/lib/postgresql/8.1/main | grep LC
LC_COLLATE: de_DE.UTF-8
LC_CTYPE: de_DE.UTF-8
```

Sollte das schiefgehen bzw. hier ein anderes Ergebnis erscheinen, ist der einfachste Weg, die Datenbank komplett neu aufzusetzen und zu initialisieren. Das tun Sie mit folgendem Befehl als User *postgres*:

```
postgres:cd /usr/lib/postgresql/8.1/bin/
postgres:./initdb -D /var/lib/postgresql/8.1/main -E
UTF8 --locale=de_DE.UTF-8
    --lc-ctype=de_DE.UTF-8
```

Mit dem Programm *pg_controldata* überprüfen wir, ob die *Locale* für unseren Datenbank-Cluster richtig gesetzt ist. Das sieht auch gut aus. Jetzt spielen wir also den Dump mithilfe von *psql* ein:

```
postgres: psql tsearch < tsearch2.sql
```

Sie sehen am Bildschirm einige Ausgaben wie *INSERT*, *CREATE TABLE* und *CREATE FUNCTION*. Ebenfalls gibt es ein paar Hinweise, die Sie aber getrost vernachlässigen können. Alles ok!

Konfiguration der tsearch2 Extension

Schauen wir uns als Nächstes an, was in unserer Datenbank steht:

```
postgres: psql tsearch
tsearch=#
tsearch=# \d
   Liste der Relationen
Schema |   Name   |  Typ  | Eigentümer
-------+----------+-------+------------
public | pg_ts_cfg   | Tabelle | postgres
public | pg_ts_cfgmap | Tabelle | postgres
public | pg_ts_dict  | Tabelle | postgres
public | pg_ts_parser | Tabelle | postgres
```

Hervorragend! Alle Tabellen der tsearch2 Extension sind vorhanden. Diese Tabellen werden herangezogen, um einen eingegebenen Text in einen *tsvector* zu verwandeln. Der Parser der Engine splittet den Text in Zeichenfolgen (Tokens) auf, und mithilfe der Wörterbücher werden die Zeichenfolgen dann in Lexeme (lexikalische Einheiten) umgewandelt. Hier eine kurze Erläuterung, was in den Tabellen steht:

pg_ts_cfg	Hier stehen die unterschiedlichen Konfigurationen für einzelne locales
pg_ts_cfgmap	Hier wird für jede Konfiguration aus pg_ts_cfg angegeben, wie mit unterschiedlichen tokens umgegangen werden soll
pg_ts_dict	Hier werden die einzelnen Wörterbücher aufgeführt
pg_ts_parser	Alle verfügbaren Parser, wobei der Default Parser für alle Text- und HTML-Dokumente völlig ausreichen sollte

Neben diesen Tabellen werden insgesamt 83 Funktionen und ein Trigger in unsere Datenbank gepackt. Es ist Zeit für den ersten Test. Eine sehr nützliche Funktion ist *ts_debug*:

```
tsearch=# SELECT * FROM ts_debug('Wir testen unsere
tsearch2-Installation in PostgreSQL 8.1');
FEHLER:  could not find tsearch config by locale
CONTEXT:  SQL-Funktion »_get_parser_from_curcfg«
Anweisung 1
SQL-Funktion »ts_debug« beim Start
```

Das sieht nicht so gut aus. Das Problem: Es gibt keine Konfiguration in der Tabelle *pg_ts_cfg*, die zu unserer Locale passt. Warum? Sehen wir uns den Inhalt der Tabelle *pg_ts_cfg* an:

```
tsearch=# SELECT * FROM pg_ts_cfg;
ts_name     | prs_name | locale
---------------+----------+--------------
default     | default | C
default_russian | default | ru_RU.KOI8-R
simple      | default |
```

Da liegt das Problem. Wir gehen einmal davon aus, dass Sie die Locale *ru_RU.KOI8-R* nicht in Ihrem Datenbankcluster einsetzen wollen. Unsere Locale ist, wie oben beschrieben, *de_DE.UTF-8*. Diese Locale müssen wir also auch in *pg_ts_cfg* eintragen:

```
tsearch=# INSERT INTO pg_ts_cfg (ts_name, prs_name,
                                                locale)
    VALUES ('default_german', 'default', 'de_DE.UTF-8');
                                                INSERT
```

Und jetzt testen wir noch einmal:

```
tsearch=# SELECT * FROM ts_debug('Wir testen unsere
                    tsearch2 Installation in PostgreSQL 8.1');
ts_name | tok_type | description | token | dict_name |
                                                tsvector
--------+----------+-------------+-------+-----------+-----
-----
(0 Zeilen)
```

Das sieht schon viel besser aus, allerdings liefert die Abfrage kein Ergebnis. Der Grund dafür ist die fehlende Verknüpfung für *default_german* in der Tabelle *pg_ts_cfgmap*, deshalb ändern wir erst einmal die aktuelle Konfiguration zu *default* – denn russisch ist unser Text wirklich nicht.

```
tsearch=# SELECT set_curcfg(‚default‘);
tsearch=# SELECT * FROM ts_debug(‘Wir testen unsere
                    tsearch2 Installation in PostgreSQL 8.1‘);
ts_name | tok_type |description   |   token   |
                                        dict_name |  tsvector
---------+-------+------------------+-------------+----------
--+--------------
 default | lword | Latin word    | Wir      | {en_stem} | ‘wir‘
 default | lword | Latin word    | testen    | {en_stem} |
                                                    ‘testen‘
 default | lword | Latin word    | unsere    | {en_stem} |
                                                    ‘unser‘
 default | word  | Word       | tsearch2   | {simple} |
                                                  ‘tsearch2‘
 default | lword | Latin word    | Installation | {en_stem}
                                                    | ‘instal‘
 default | lword | Latin word    | in      | {en_stem} |
 default | lword | Latin word    | PostgreSQL  | {en_stem}
                                                  | ‘postgresql‘
 default | float | Decimal notation | 8.1     | {simple} | ‘8.1‘
(8 Zeilen)
```

Alles klappt hervorragend. Doch in der Spalte *dict_name* sehen Sie, welche Wörterbücher genutzt werden: Einmal *en_stem*, was bedeutet, das Stemmer Wörterbuch *englisch* wird für alle *Latin*-Alphabet-Worte genutzt, und ein ganz einfaches Wörterbuch *simple*, wobei hier keine Stopwörter verworfen werden. Wir wollen aber bitte ein deutsches Wörterbuch nutzen – in unserem Fall *ispell*. Dazu müssen einige Anpassungen vorgenommen werden, die wir hier aus Platzgründen weniger ausführlich beschreiben. Gehen Sie bitte folgendermaßen vor:

- Stellen Sie sicher, dass *ispell* auf Ihrem Rechner installiert ist.
- Im Verzeichnis */usr/lib/ispell* müssen folgende Dateien vorhanden sein: *english.aff*, *english.dict*, *english.stop german.aff*, *german.med*, *german.stop*. *ispell*.
- Im Verzeichnis */usr/share/postgresql/8.1/contrib* legen Sie die Datei *german.stop* ab.
- Die Generierung der *german.**-Dateien wird Ihnen unter [4] erklärt und die *english**-Dateien können Sie unter [5] herunterladen.
- Integrieren Sie die englische Sprache in *pg_ts_cfg*:

```
INSERT INTO pg_ts_cfg (ts_name, prs_name, lo cale)
VALUES (‘default_german‘, ‘default‘, ‘de_DE.UTF-8‘);
```

- Erweitern Sie die Tabelle *pg_ts_cfgmap* mit dem Inhalt des SQL Dumps, den Sie unter [2] im Bereich *Download* finden.
- Wiederholen Sie diese Inserts für die Sprache Englisch wobei Sie entsprechend *default_english* und *{en_ispell, en_stem}* verwenden.
- Und zu guter Letzt fügen Sie diese Einträge in die Tabelle *pg_ts_dict* ein:

```
INSERT INTO pg_ts_dict(SELECT de_ispell‘,dict_init,
    ‘DictFile=“/usr/lib/ispell/german.med“,‘
    ‘AffFile=“/usr/lib/ispell/german.aff“,‘
    ‘StopFile=“/usr/lib/ispell/german.stop.ispell“‘,
    dict_lexize
      FROM pg_ts_dict
      WHERE dict_name = ‘ispell_template‘);
```

```
INSERT INTO pg_ts_dict (dict_name, dict_init, dict_initoption, dict_lexize, dict_comment)
VALUES ('de_stem', 'snb_en_init(internal)', 'contrib/german.stop', 'snb_le xize(internal,internal,inte ger)', 'German Stemmer Stop');
```

Wiederholen Sie die beiden Inserts mit den entsprechenden Einträgen für die Sprache Englisch. Jetzt noch einmal unser Test in Deutsch:

```
tsearch=# SELECT set_curcfg('default_german');
tsearch=# SELECT * FROM ts_debug('Wir testen unsere tsearch2-Installation in PostgreSQL 8.1');
ts_name | tok_type | description| token | dict_name | tsvector --------+----------+--------------+--------+-----------+---------------
default_german | lword |Latin word| Wir | {de_ispell,de_stem}|
default_german | lword |Latin word| testen | {de_ispell,de_stem}| 'testen' default_german | lword |Latin word| unsere | {de_ispell,de_stem}| 'seren' 'unser'
default_german | word |Word |tsearch2|{de_ispell,de_stem} | 'tsearch2'
default_german | lword |Latin word|Installation|{de_ispell,de_stem}|'installation' default_german | lword |Latin word| in | {de_ispell,de_stem} |
default_german | lword |Latin word| PostgreSQL|{de_ispell,de_stem}|'postgresql'
default_german | float |Decimal notation| 8.1|{simple} | '8.1'
(8 Zeilen)
```

Jetzt sind wir am Ziel angelangt. Schauen wir uns das Ergebnis an. In *dict_name* sehen wir, welche Wörterbücher für diesen Token genutzt wurden. Da in der ersten Zeile in *tsvector* kein Eintrag steht, wurde das Wort *Wir* als Stoppwort erkannt – genauso wie das Wort *in*. Prima! Das Wort *unsere* wurde auf Grund des *ispell*-Wörterbuchs sogar in zwei Vektoren aufgesplittet – naja, in nicht wirklich sinnvolle. Und zum Schluss sehen wir noch einen Token-Typ *float*, der als Zahl erkannt wurde.

Sie haben jetzt die tsearch2 Engine installiert, eingerichtet und auf Ihre richtige Funktionsweise hin getestet. Dann lassen Sie uns ans Kodieren gehen – wir bauen eine Suchmaschine.

Eine Suchmaschine mit der tesearch2 Extension

Am einfachsten lässt sich eine Suchmaschine beschreiben, wenn man mit ihr suchen kann. Klingt logisch – oder? Wir haben eine Beispielapplikation für Sie erstellt. Machen Sie es sich ein wenig einfacher und laden Sie die Applikation unter [7] herunter. Entpacken Sie das Zip-Archiv in ein Verzeichnis, auf das der Webserver Zugriff hat. Voraussetzung für die Nutzung ist natürlich eine korrekt installierte PostgreSQL mit der tsearch2-Erweiterung. Sie können auf der Startseite der Beispielapplikation testen, ob diese Voraussetzung erfüllt ist. Klicken Sie dazu einfach auf den Button *Testabfrage starten*.

Alternativ können Sie sich die Applikation aber auch zuerst unter [2] ansehen. Geben Sie unter *Daten eingeben* mehrere Datensätze ein. Dabei sollten Sie bestimmte Keywords in die Felder *Titel*, *URL* und *Text* einfügen. Sie werden dann beim Durchsuchen der Datenbank unter *Suchen* in den vollen Genuss des Rankings kommen.

Dokumentinhalte in die Datenbank schreiben

Wir haben eingangs beschrieben, dass mit der Funktion *to_tsvector* Vektoren aus den einzelnen relevanten Worten eines Textes erzeugt und in einer Spalte (hier *vector*) gespeichert werden. Dabei haben Sie auch die Möglichkeit zu bestimmen, wie gewisse Informationen aus unterschiedlichen Quellen gewichtet werden sollen. Dafür stehen die Buchstaben A bis D zur Verfügung, wobei A die höchste und D die geringste Gewichtung bezeichnet.

Mit unterschiedlichen Quellen sind in unserem Fall schlicht unterschiedliche Formularfelder. Ein Webcrawler einer Suchmaschine gewichtet unterschiedliche Bereiche und Informationen einer Website ebenfalls je nach Relevanz. So wird z.B. eine Überschrift in <h1> Tags höher gewichtet als Text (bzw. Worte aus diesem Text) in einem Bereich weiter unten auf einer Website („Suchmaschinenoptimierung“, Sebastian Erlhofer). Zurück zu unserer Applikation.

Zuerst müssen wir eine Tabelle erstellen, in der wir unsere Daten vorhalten werden. Der User, den wir nutzen heißt *volltextsucher*:

```
CREATE TABLE documents (
 id serial NOT NULL,
 title character varying(255),
 url character varying(255),
 body text,
 author character varying(255),
 vector tsvector,
 CONSTRAINT documents_pkey PRIMARY KEY (id)
)
WITHOUT OIDS;
ALTER TABLE documents OWNER TO volltextsucher;
```

Beachten Sie, dass das Feld *vector* vom Typ *tsvector* ist. Das ist ein eigener Datentyp der tsearch2 Engine und ermöglicht das Speichern von Vektoren in dieser Spalte. Im Anschluss daran erstellen wir noch den Index auf die Spalte *vector*:

```
CREATE INDEX docs_index ON documents USING gist(vector);
```

Die Erzeugung des Index ist wichtig, da die Suche in einer indexierten (also sortierten) Spalte wesentlich schneller ist als in einer nicht indexierten Spalte. Der *gist()*-Index ist eine Art von Index, der auf die Verwendung in der tsearch2 Engine optimiert ist. Sie müssen nicht zwangsläufig einen Index einsetzen, wir raten Ihnen jedoch dringend dazu.

In unserem Beispiel gibt es vier Informationsquellen, und somit vier Formularfelder mit entsprechender Gewichtung:

Titel → Gewichtung A
URL → Gewichtung B
Text → Gewichtung C
Autor → Gewichtung D

Wir schreiben die Daten folgendermaßen in die Datenbank:

```
$qs = INSERT INTO documents (title, url, body, author, vector)
  VALUES ( '".$params['title']."', '".$params['url']."',
    '".$params['body']."', '".$params['author']."',
setweight(to_tsvector('".$params['title']."'), 'A') ||
setweight(to_tsvector('".$params['url']."'), 'B') ||
setweight(to_tsvector('".$params['body']."'), 'C') ||
setweight(to_tsvector('".$params['author']."'), 'D') )";
$db->sql_add($qs);
$db->transaction();
```

Was tun wir hier? Wir schreiben die Formulardaten der Felder *title*, *url*, *body* und *author* in die entsprechenden Spalten der Tabelle *documents*. Zusätzlich schreiben wir alle Formulardaten noch einmal zusammen in die Spalte *vector*, wobei wir zum einen die Gewichtung bestimmen und zum anderen die Funk-

tion *to_tsvector* nutzen, um die Daten aufbereitet und als Vektor abzulegen. Beispiel:

```
INSERT INTO documents (title, url, body, author, vector)
VALUES ('Volltext Suche mit tsearch2 und PostgreSQL - PHP
  Magazin 5.08',
  'http://it-republik.de/php/',
    'Um einigermaßen sinnvolle Suchergebnisse zu
      erhalten, müssen wir im ersten Schritt Daten in die
      Datenbank schieben. Dafür wird die tsearch2-
      Funktion to_tsvector genutzt. Gleichzeitig
      bestimmen wir mit der Funktion setweight und den
      Buchstaben A - D die Gewichtung. Dabei ist A die
      höchste und D die niedrigste Gewichtung.',
    'Andreas Wenk und Thomas Pfeiffer',
    setweight(to_tsvector('Volltext Suche mit tsearch2
      und PostgreSQL - PHP Magazin 5.08'), 'A') ||
      setweight(to_tsvector('http://it-republik.de/php/'),
      'B') ||
      setweight(to_tsvector('Um einigermaßen sinnvolle
        Suchergebnisse zu erhalten, müssen wir im ersten
        Schritt Daten in die Datenbank schieben. Dafür
        wird die tsearch2-Funktion to_tsvector genutzt.
        Gleichzeitig bestimmen wir mit der Funktion
        setweight und den Buchstaben A - D die
        Gewichtung. Dabei ist A die höchste und D die
        niedrigste Gewichtung.'), 'C') ||
      setweight(to_tsvector('Andreas Wenk und Thomas
        Pfeiffer'), 'D')
  )
```

Wenn wir danach eine Abfrage an die Tabelle *documents* starten, erhalten wir folgendes Ergebnis der Spalte *vector*:

```
'a':47C,53C 'd':48C,57C 'to':34C 'php':7A '5.08':9A
'wenk':62 '/php/':12B 'dabei':51C 'daten':24C 'and-
rea':61 'dafür':29C 'ersten':22C 'suchen':2A 'thomas':64
'genutzt':36C 'gewicht':50C,60C 'magazin':8A 'müssen':19C
'niedrig':59C 'schritt':23C 'erhalten':18C 'funktion':33C,42C
'höchste':55C 'pfeiffer':65 'schieben':28C 'sinnvoll':15C
'tsearch2':4A,32C 'tsvector':35C 'volltext':1A
'bestimmen':38C 'buchstabe':46C 'datenbank':27C
'setweight':43C 'buchstaben':46C 'postgresql':6A
'gleichzeitig':37C 'einigermassen':14C 'suchergebniss':16C
'it-republik.de':11B 'it-republik.de/php/':10B
```

Wir sehen hier, wie tesearch2 zum einen den eingegebenen Text reduziert und Informationen zu den einzelnen Worten als Vektor abgelegt hat. Das ist die Position des Worts im Dokument und deren Gewichtung. Diese Informationen sind jetzt die Grundlage für unsere spätere Suche.

Ein Hinweis zum Speichern von Dokumenten per *INSERT* sei noch gegeben: In einem produktiven System würden Sie eine Funktion (function, stored procedure) für das Speichern der Dokumente und deren Inhalte schreiben[6].

Die Dokumente durchsuchen

Nachdem wir nun unsere Dokumentinhalte in die Datenbank geschrieben haben, können wir sie durchsuchen. Dafür geben wir die Inhalte nicht einfach aus, sondern möchten die Ergebnisse, die am besten zu unserer Suchanfrage passen, ganz oben in der Ergebnisliste erhalten. Außerdem möchten wir einen Ausschnitt aus dem Inhalt im Suchergebnis erhalten. Hier die entsprechende Abfrage:

```
$qs = " SELECT id, headline(title, q) as headline,
  headline(body, q) as body,   rank(vector, q)
  FROM documents, to_tsquery('".$searchPhrase."') AS q
  WHERE vector @@ q
  ORDER BY rank(vector, q) DESC ";
$res = $db->select($qs);
```

Gehen wir das SQL-Statement durch: Mit der Funktion *headline()* geben wir an, dass die Worte, die unserer Suchanfrage entsprechen, gesondert formatiert dargestellt werden sollen. In der Standardeinstellung ist dies einfach fett (<*b*>...</*b*>). Sie können auch mehrere Spalten als *headline* ausgeben. Achten Sie aber darauf, dass Sie dann die jeweilige *as irgendwas*-SQL-Notation nutzen.

rank() gibt an, dass wir ein Ranking auf die Ergebnisse der Spalte *vector* anwenden wollen. Grundlage bildet *q*, was letztlich eine Abkürzung für *to_tsquery* (*[SUCHE]*) ist. Die Funktion *to_tsquery* in der *FROM*-Anweisung wandelt die Suchanfrage in einen tsearch2-Ausdruck mit allen Funktionalitäten um. Ist das erfolgt, kann dieser Ausdruck auf die Spalte *vector* mit unseren ganzen Vektoren losgelassen werden. Der tsearch2-eigene Operator @@ sagt, dass wir eine Volltextsuche durchführen wollen. Ganz am Ende nehmen wir dann noch die Sortierung vor, indem wir wieder die Funktion *rank()* nutzen und ein Ranking der Einträge – in unserem Fall absteigend – vornehmen. Das war's!

Als Ergebnis erhalten Sie nun eine einfache Liste mit gefundenen Einträgen, die zu unserer Suchanfrage am besten passten. Wenn Sie die Beispielapplikation installiert haben oder sich diese unter [2] ansehen, können Sie das Ranking ganz klar erkennen. Dabei ist wichtig zu wissen, dass das Ranking intern die Gewichtung der einzelnen Worte (A, B, C, D) in Kombination mit der Häufigkeit des Vorkommens der Worte im Dokument berücksichtigt.

Fazit

Wir haben Ihnen im vorliegenden Artikel die grundlegende Funktionsweise der PostgreSQL Extension tsearch2 vorgestellt. Natürlich ist die Beispielapplikation sehr rudimentär. Trotzdem können Sie schon mit dieser einfachen Implementierung erstaunliche Ergebnisse generieren. Es sei auch noch erwähnt, dass die Extension auch eine Funktionalität für Thesaurus und Synonyme vorsieht. Dafür müssen entsprechende Wörterbücher integriert werden. Ist tsearch2 nicht noch ein Grund, zukünftig die PostgreSQL-Datenbank einzusetzen? Wir laden Sie ein, in Ihren Projekten eine Suchmaschine auf Basis der tsearch2 Extension und der PostgreSQL-Datenbank zu integrieren.

Links & Literatur

[1] postgreSQL Textsearch Doku in PostgreSQL 8.3: http://www.postgresql.org/docs/8.3/static/textsearch-intro.html

[2] Beispielapplikation online: http://phpmag.e-unique.com/

[3] pgAdmin: http://www.pgadmin.org

[4] german.*: http://www.sai.msu.su/~megera/postgres/gist/tsearch/V2/docs/tsearch2_german_utf8.html

[5] english*: http://www.sai.msu.su/~megera/postgres/gist/tsearch/V2/

[6] tsearch2: http://www.sai.msu.su/~megera/postgres/gist/tsearch/V2/docs/tsearch2-guide.html

[7] Download Beispielapplikation: http://phpmag.e-unique.com/downloads/tsearch2_applikation_phpmag.e-unique.com.zip

postgreSQL: http://www.postgresql.org

THOMAS PFEIFFER & ANDREAS WENK

Thomas Pfeiffer und Andreas Wenk sind Applikationsentwickler bei der NMMN – New Media Markets & Networks GmbH in Hamburg. Dort entwickeln sie das Ressourcen Management System eUNIQUE unter heftigem Einsatz von PHP und der PostgreSQL. Sie erreichen die beiden unter tp@nmmn.com und aw@nmmn.com oder über die Webpräsenz http://www.e-unique.com.