www-Suchmaschinen als Quellen für Datenbankanfragen

Thorsten Tillack

6. Februar 2006

Zusammenfassung

www-Suchmaschinen sind in zunehmendem Maße für Web Service-Anfragen zugänglich.
Gleichzeitig finden sich immer neue Anwendungen, die diese Web Service-Anfragen nicht primär als klassische Information Retrival-Anfrage nutzen, sondern als Datenquellen.
Inwiefern kann man die Datenquelle Internet zur Verifikation von Zusammenhängen nutzen?
Kann man das heutige Internet wie eine *normale* Datenbank ansprechen und verwenden?
Können diese *Datenbankanfragen* in anderen Datenbanken zur Verifikation dienen?
Wie sicher sind die ermittelten Aussagen? Gibt es Grenzen?
Antwort: Ja! Man kann das Internet als Datenbank zur Verifikation von anderen Daten nutzen und das Ergebnis anderen Datenbanken bereitstellen. Aber wie können die Datenbanken auf das Internet zugreifen und welche Einschränkungen gibt es? Diese Fragen sollen im Folgenden geklärt werden.

Inhaltsverzeichnis

1 Eine Einführung **3**

2 Was bisher geschah... **4**
2.1 PANKOW 4
2.2 C-PANKOW 5
2.3 Fehlende Funktionalität 5
2.4 Das Szenario 6

3 Technische Vorraussetzungen **7**
3.1 Die GoogleAPI 7
3.2 Postgre 7
3.3 Table Functions 8
3.4 Die prozeduralen Sprachen von postgre 8
3.5 plperl / plperlU 9

4 google.pl - Das Perl-Skript **11**
4.1 Die eigentliche Anfrage 12

5 Die Table Function googleJoin(..) **14**
5.1 Der Grundgedanke der Funktion 14
5.2 Die Implementierung 15
5.3 Die Tabellen employee und fb 18
5.4 Beispiel-Anfragen und Ergebnisse 18
5.5 Weitere Anfrage-Beispiele 19

6 Wieso keine fixe Grenze? **21**

7 Bewertung der Funktion googleJoin(..) **22**

8 Verbesserung von googleJoin **23**
8.1 Hierarchisierung der Treffer 23
8.2 Eine erste Annäherung 24
8.3 C-PANKOW meets googleJoin 25

9 Fazit **28**
9.1 Der normale googleJoin 28
9.2 Der Kontext-googleJoin 29

10 Literatur **32**

11 Quelltexte **33**
11.1 googleJoin() 33
11.2 googleJoinC() 36

1 Eine Einführung

In der heutigen Zeit nimmt die Bedeutung von Datenbanken im alltäglichen Leben immer mehr zu und das Internet ist aus dem normalen Leben gar nicht mehr wegzudenken.
Außerdem nimmt die Abhängigkeit von Datenbanken und dem Internet untereinander immer größere Ausmaße an.
Es ist heutzutage ganz normal, dass auf einer Webseite ein Gästebuch für die Besucher vorhanden ist. Diese Einträge werden dann in einer Datenbank, meist mySQL, gespeichert. Wenn man sich dann diese Einträge ansehen möchte, muss man mittels der Anfragesprache SQL wieder auf die Datenbank zugreifen und bekommt das Ergebnis der Anfrage auf der Internetseite präsentiert.
Hier bekommt die Datenbank ihre Informationen von aussen eingegeben und muss sich nicht um das Beziehen der Daten kümmern. Wieso aber lädt sich die Datenbank nicht die Informationen (z.B. über das Wetter auf Hawaii) selbst aus dem Netz?!
Was spricht dagegen, das Netz als Informationsquelle, als Ansammlung menschlichen Wissens, als Datenbank zu nutzen?
Eine Datenbank, die von einer anderen Datenbank nach Informationen durchforstet wird, welche sich dann ihre Ergebnisse selbst zusammensucht.

Informationen mittels Semi-Automatismen aus dem Netz zu extrahieren, diese Ergebnisse dann auf einzelnen Internet-Seiten zu präsentieren ist seit der Entwicklung der GoogleAPI[1] sehr einfach möglich. So gibt es viele Seiten, die eine Suchmöglichkeit anbieten, welche dann die Anfrage an Google weiterleitet und dessen Ergebnisse wieder auf der Ausgangsseite anzeigt.
Funktioniert dieser Mechanismus auch auf Datenbankebene?
Die Google-Anfragen aus einer Datenbank herauszustellen dürfte kein Problem sein, aber kann man die Ergebnisse auch mittels einer Datenbank verarbeiten und damit z.B. Beziehungen zwischen einzelnen Objekten ermitteln?
Diese Fragestellung soll hier nun überprüft werden...

[1] vgl. Kapitel 3.1

2 Was bisher geschah...

Es ist keine neue Idee das Internet als Wissensbasis zu nutzen.
Das so entstehende semantische Netz wird auf Grund von Metadaten der Internetseiten in eine Ontologie eingegliedert.
Die bisherigen Methoden das Internet als Informationsbasis zu nutzen basierten auf der Zerlegung von Webseiten. Hierzu wurde den einzelnen Verfahren eine Internetseite eingegeben, dessen html-Text anschließend um die html-Tags bereinigt und analysiert wurde.
Bei PANKOW[2] wird eine Instanz mittels verschiedener Sprachfetzen zu Phrasen zusammengefügt und auf ihre sematische Bedeutung hin, mit Hilfe des Internets, überprüft.
Diese Methode war schon recht vielversprechend, hat jedoch einen erheblichen Overhead an *unnötigen* Phrasen und somit zu einem übermäßigen Datenverkehr geführt. Diese *Nachteile* wurden mit C-PANKOW[3] ausgeräumt.

Bei C-PANKOW werden zuerst die Inhaltsangaben der Seite aus dem Internet heruntergeladen und off-line verarbeitet. Diese Inhalte werden linguistisch untersucht und normalisiert, wodurch ein erhöhter Abdeckungsgrad der erstellten Phrasen erzeugt wird.

2.1 PANKOW

PANKOW steht für Pattern-based Annotation through Knowledge on the Web und damit für eine musterbasierende Erläuterung durch Wissen im Netz, was einen unüberwachten Mechanismus beschreibt, der ein Nomen (im Folgenden auch: Instanz) in eine Ontologie einordnen soll.
PANKOW bedient sich hierbei folgenden Verfahrens:

- PANKOW wird eine Webseite übergeben, dessen Nomen in einen Kontext zu einem vorher gegebenen Konzept zu bringen sind.
  Beispielsweise soll überprüft werden, ob es sich bei den Nomen um ein Land oder Hotel handelt.
- Diese Nomen werden nun mittels Sprachmustern, z.B. *...ist ein...*, mit den Konzepten zu Phrasen zusammengebaut und als Anfrage an Google weitergeleitet.
- Die GoogleAPI liefert nun eine Trefferanzahl zu den einzelnen Phrasen.
- Anhand der Trefferanzahl werden nun die Phrasen (bzw. die Nomen-Konzept-Paare) in eine Reihenfolge gebracht und das jeweilige meistgefundene Nomen-Konzept-Paar wird als die *wahrscheinlichste* Relation der Objekte angesehen.

Hierdurch erhalten wir ontologische Erläuterung der Webseite[4].

[2] vgl. Kapitel 2.1
[3] vgl. Kapitel 2.2
[4] vgl. [www04]

2.2 C-PANKOW

Bei C-PANKOW handelt es sich um den jüngeren Bruder von PANKOW, der dessen Schwächen reduzieren bzw. vermeiden soll.
Eine Schwäche von PANKOW war der Mechanismus der Mustererkennung, da einige Instanzen der Sprachmuster in komplexen Satzkonstrukten nicht korrekt verarbeitet werden konnten.
Außerdem bestand die Schwierigkeit den Plural der gefundenen Nomen korrekt zu bilden, was eine Verfälschung der Suchanfragen und somit der Ergebnisse nach sich zog.
Dieses Problem wurde behoben, indem man die Seite erst heruntergeladen und anschließend sprachlich untersucht hat.
Mittels dieser Vorverarbeitung wurden gleichzeitig irrelevante Anfragen an Google und somit unnötiger Datenverkehr reduziert.
Eine dritte Schwäche von PANKOW war, dass die Anzahl der Anfragen an Google sich proportional zur gegebenen Ontologie verhielt, also eine minimale Vergrößerung der Ontologie zu einer wesentlichen Vermehrung der Anfragen führte. In C-PANKOW wird nur eine konstante Anzahl von Anfragen an Google gesendet, mittels welcher die Erläuterungen erzeugt werden. Aus diesem Grund ist C-PANKOW im Vergleich zu PANKOW eher in der Lage große Ontologien zu bearbeiten ohne dass es zu einem rapiden Anstieg an Anfrage und somit Datenverkehr kommt.
Mit diesen Erweiterungen erfolgt C-PANKOW[5] nun folgendermaßen:

1. Die zu untersuchende Seite wird zuerst nach Nomen gescannt.
2. Anschließend wird für jedes Nomen-Muster-Paar eine Anfrage an Google gesendet und die Inhaltsangaben der ersten **n** Treffer heruntergeladen.
3. Nun wird die Ähnlichkeit zwischen der zu untersuchenden Seite und den Inhaltsangaben ermittelt. Übersteigt diese Ähnlichkeit einen gegebenen Grenzwert, so beschreibt das aktuelle Muster eine Phrase, die wahrscheinlich das Konzept beschreibt, zu der das Nomen gehört.
4. Hiernach wird das Ergebnis pro Nomen nun anhand der errechnete Ähnlichkeiten aktualisiert. Hierbei wiegen Muster, die in sehr ähnlichen Inhaltsangaben gefunden werden mehr als weniger ähnliche Treffer.
5. Zum Abschluss wird das Nomen mit dem Konzept erläutert, welches die höchste Trefferanzahl hat.

Durch dieses Verfahren erhält man mindestens genauso zuverlässige Ergebnisse wie mit dem *normalen* PANKOW, allerdings mit wesentlich weniger Datenverkehr, aber mit einer Zunahme an benötigter Rechenleistung.

2.3 Fehlende Funktionalität

In Kapitel 2.1 und 2.2 werden Verfahren vorgestellt, die Informationen aus dem Internet extrahieren. Diese Verfahren ermöglichen allerdings in dieser Form nicht das Verarbeiten von Informationen aus Datenbanken, welche mit Hilfe des Internets bzw. genauer der GoogleAPI verifiziert werden sollen.
Diese Funktionalität soll nun in eine postgre-Datenbank eingefügt werden.

[5] vgl. [www05]

2.4 Das Szenario

Angenommen es bestehen zwei Tabellen. Eine mit den Namen von Mitarbeiter und eine mit den Namen der Fachbereiche der Universität Koblenz.
Es soll nun mittels dieser Tabellen die Zugehörigkeit der einzelnen Mitarbeiter zu den einzelnen Fachbereichen ermittelt werden. Mit den alten PANKOW- und C-PANKOW-Filtern müsste man hierzu die Seiten der Universität durchsuchen um alle Mitarbeiter zu lokalisieren, wobei hier das Finden der Namen schwerer sein wird als das Finden der zu erklärenden Nomen.
Nun müßte man ähnlich mit den Namen der einzelnen Fachbereichen verfahren. Diese beiden Ergebnismengen müßten nun paarweise mittels Sprachmustern wieder zu Phrasen verbunden werden und diese Phrasen anschließend je nach Mechanismus verarbeitet werden.
Dieses Vorgehen ist allerdings bei der Verwendung von Datenbanken schwierig, da Datenbanken nicht ohne weiteres über das Internet auslesbar sind.
Genau diese Verwendung von Datenbanken in Verbindung mit der direkten Bewertung der einzelnen Anfragen mittels dem Internet soll nun untersucht werden.
Vorteil dieser Methode ist eine Reduzierung des Aufwandes, sei es in der Rechenzeit oder im Datenaufkommen und somit in der Anzahl der Anfragen. Diese Vorteile entstehen, da hier direkt die Anfragen erzeugt und abgesetzt werden können. Es muss also nicht der Umweg über das Zerlegen der einzelnen Internetseiten gemacht werden.
Als Ergebnis wird eine Aussage über den *dynamischen Join*[6] der beiden Tabellen überliefert. Das bedeutet, dass wenn genug Treffer im Internet zu der Mitarbeiter-Fachbereich-Konstellation gefunden werden 'true', andernfalls 'false' zurückgegeben wird.

[6] vgl. Kapitel 3.3

3 Technische Vorraussetzungen

Um dieses Verfahren, was man *T-PANKOW* (Table-oriented Pattern-based Annotation through Knowledge on the Web) nennen könnte, verwenden zu können, werden einige technische Vorraussetzungen an die Datenbank bzw. an den ausführenden Computer gerichtet.

3.1 Die GoogleAPI

Um Informationen aus dem Netz beziehen zu können muss ein Information Retrieval erfolgen.
Das (semi-)automatische Information Retrieval über das Internet war meist recht aufwändig. Man musste aus dem Programm eine Internetseite öffnen, hier dann ggf. eine Anfrage an die Suchmaschinen absetzten und anschliessend die Ergebnisseite anhand der html-Tags parsen um das gewünschte Ergebnis ins Programm zu übernehmen.
Von Google wird seit dem 30.08.2002 eine API zur direkten Nutzung der Suchmaschine als Web Service bereitgestellt.
Durch diese API lassen sich die Anfragen direkt aus den Programmen heraus generieren, abschicken und das Ergebnis wird auch direkt wieder in das Programm zurückgeliefert. Hierdurch wird das Information Retrieval stark vereinfacht.
Die beta-Version des Web Service kann in die gängigen Programmiersprachen eingebettet werden: Java, C, C++, Perl sowie alle .NET-Sprachen. Hierzu werden entweder zu inkludierende Pakete bereitgestellt oder eine wsdl-Datei, die für alle SOAP-unterstützenden Sprachen, z.B. Perl, gedacht ist. Zur Nutzung der API muss bei www.google.com/apis ein Konto erstellt werden, da jedem Nutzer der API anfangs nur ein Kontingent von 1000 Anfragen pro Tag zur Verfügung steht. Nach Erstellung eines Kontos bekommt man von Google einen Key geliefert, welchen man bei der Anfrage angeben muss.

3.2 Postgre

Da der *dynamische Join* aus Tabellenwerten erzeugt werden soll, bietet es sich an, das Ergebnis auch wieder in einer Tabelle bzw. Spalte darzustellen, mit der man dann direkt auf der Ausgangsdatenbank weiterarbeiten kann.
Um diese Fähigkeit zu nutzen benötigt man sogenannte *Table Functions*. Diese Table Functions werden nicht von jedem Datenbanksystem unterstützt. So unterstützt z.B. mySQL diese Functions nicht, postgre allerdings schon, weshalb die Wahl des Datenbanksystem recht schnell und einfach auf postgre fällt.

Bei postgre handelt es sich um ein leistungsstarkes open-source-Datenbanksystem, das es neben den normalen Datenoperation und Anfragen ermöglicht virtuelle Tabellen mittels Table Functions zu definieren. Diese virtuellen Tabellen können dann, wie jede andere Tabelle, angesprochen und genutzt werden. Hierdurch erhöhen sich natürlich automatisch die Fähigkeiten und der Umfang der Einsatzmöglichkeiten dieses Datenbanksystems.

Das eigentliche Interface steht dem Administrator über die Konsole zur Verfügung. Es wird mittels 'psql DBNAME' aufgerufen.

3.3 Table Functions

Wie schon in Kapitel 3.2 erwähnt wird die Fähigkeit des Erzeugens von Table Functions benötigt.
Diese Table Functions bezeichnen die Möglichkeit virtuelle Tabellen zu definieren. In diesen nicht-real-existierenden Tabellen stehen keine fixen Werte, sondern eine Funktion in einer von postgre unterstützen Sprache. Durch diese Funktionen werden bei jedem Aufruf der Table Function der Inhalt der virtuellen Tabelle dynamisch berechnet bzw. erzeugt. Deshalb wird in unserem Szenario [7] *eine Aussage über den dynamischen Join der beiden Tabellen* als Ergebnis des Aufrufes erhalten.
Hier ein Beispiel zur Verwendung von Table Functions:

```
CREATE TABLE foo (fooid int, fooname text);
INSERT INTO foo VALUES (1, 'Joe');
INSERT INTO foo VALUES (1, 'Ed');
INSERT INTO foo VALUES (2, 'Mary');

CREATE FUNCTION getfoo(int) RETURNS foo AS '
  SELECT * FROM foo WHERE fooid = $1;
' LANGUAGE SQL;

SELECT * FROM getfoo(1);
fooid | fooname
------+--------
  1   |  Joe
(1 row)
```

Diese Beispiel[8] fragt nun mittels der Funktion getfoo(int) in der Tabelle foo nach allen Zeilen, die in der Zelle fooid den übergebenen Wert aufweisen.
Es handelt sich hierbei also um eine Schreibabkürzung für:

```
SELECT * FROM foo WHERE fooid = <?>;
```

Dies ist allerdings nur ein kleines, beschränktes Beispiel für die wesentlich mächtigeren Fähigkeiten der Table Functions.
Außerdem ist zu beachten, dass man zwar mittels einer Table Function Daten in eine Tabelle eintragen, aber keine Werte in eine Table Function eintragen kann.

3.4 Die prozeduralen Sprachen von postgre

Um die oben angesprochenen Table Functions zu realisieren bietet das postgre-System eine Reihen von prozeduralen Sprachen an, die alle von *Muttersprachen* abgeleitet werden und einen ähnlichen Funktionsumfang wie ihre Ausgangssprachen haben.

[7] Kapitel 2.4
[8] vgl. [postgre74]

Diese prozeduralen Sprachdialekte stehen dem Datenbankadministrator auf der Serverseite zur Verfügung. Der Nutzer der Table Functions hat also nicht die Möglichkeit diese zu modifizieren, wie es bei den *normalen* Anfrage-Sprachen auf der Client-Seite der Fall ist.
Allerdings wird diesen postgre-Sprachen in der Regel ein Sicherheitsriegel vorgeschoben, der nur eine Ausführung von lokalen Operationen erlaubt. Außerdem ist es nicht ohne weiteres möglich globale Variablen zwischen einzelnen Funktionsdurchläufen bzw. einzelnen Funktionen zu übermitteln oder beizubehalten.
Eine weitere Restriktion der Sprachen ist der Umfang des Ergebnisses durch die Funktionen. Diese Funktionen können nur eine einzige Spalte und keine gesamte Zeile zurückliefern. Eine kleine Verbesserung ist bei postgre Version 8.1 die Möglichkeit Mengen oder Arrays bzw. selbstdefinierte Datentypen zurückzugeben; allerdings werden diese Datentypen immernoch in einer Spalte geliefert.
Zur Erstellung der Table Functions werden standardmäßig libpq - eine C-Bibliothek, pgtcl - eine TCL Binding Bibliothek, ECPG - embedded SQL in C und das JDBC Interface auf der Client-Seite, sowie PL/TcL, PL/Python, PL/Perl mitgeliefert auf der Server-Seite, die aber aus Sicherheitsgründen auf jeder Datenbank erst aktiviert werden müssen.
Außerdem ist es zusätzlich noch möglich PL/Java, einen Java-Dialekt, nachträglich der Datenbank *beizubringen.* Hierzu müssen aber mehrere Pakete unter Linux nachinstalliert werden, was nicht ohne weiteres möglich ist und zu Problemen geführt hat.
Da im Sprachenumfang von postgre ein Perl-Dialekt enthalten ist, wurde dieser verwendet, da er durch die GoogleAPI direkt unterstützt wird, er nicht nachträglich installiert werden muss und er in den einzelnen Versionen der Datenbank unverändert übernommen werden kann, wodurch eine Aufwärts- und Abwärtskompatibilität bis jetzt gegeben ist.
Zur Verwendung von PL/Perl muss nur mittels *createlang plperl DBNAME* die Sprache über die Konsole[9] (also nicht innerhalb von 'psql') aktiviert werden.

```
CREATE FUNCTION perl_max (integer, integer) RETURNS integer AS '
  if ($_[0] > $_[1]) { return $_[0]; }
  return $_[1];
' LANGUAGE plperl;

SELECT perl_max(2,3);
perl_max
--------
   3
(1 row)
```

Dieser Beispielfunktion[10] in PL/Perl werden 2 Ganzzahlwerte übergeben und der größere wird als Ergebnis zurückgeliefert.

3.5 plperl / plperlU

Wie in Kapitel 3.4 schon angedeutet wird allen Sprachen nur eine lokale Ausführungsberechtigung erteilt. So ergeht es auch dem *normalen* PL/Perl. Dieser Sprache

[9] Anmerkung: Dies bezieht sich auf Linux-Computer!
[10] vgl. [plperl74]

wird in der trusted-Variante kein Schreibzugriff auf Dateien und, was hier wesentlich problematischer ist, keine Verbindung zu anderen Servern gestattet, selbst unter Administratorrechten nicht.
Hierdurch wird das Einbinden des, wie wir später sehen werden, benötigten SOAP-Paketes unmöglich.
Um dieses Paket einzubinden muss der *böse* Zwilling von PL/Perl herangezogen werden. In der *untrusted* Version von PL/Perl, PL/PerlU, ist keinerlei Sicherheitsriegel eingebaut und der Programmierer muss selbst für das Handeln seiner Arbeit die Verantwortung und Sicherung übernehmen.
Dies ermöglicht das Benutzen des SOAP-Paketes und die Verbindung zum externen Google-Server.
Zur Verwendung von PL/PerlU muss dementsprechend mittels *createlang plperlu DBNAME* die Sprache aktiviert werden, wobei die Aktivität von PL/PerlU keine Benutzung der Sprache PL/Perl voraussetzt.

```
CREATE FUNCTION badfunc() RETURNS integer AS '
  my $tmpfile = "/tmp/badfile";
  open my $fh, '>', $tmpfile
    or elog(ERROR, qq{Could not open the file "$tmpfile": $!});
  print $fh "Testing writing to a file\n";
  close $fh or
        elog(ERROR, qq{Could not close the file "$tmpfile": $!});
  return 1;
' LANGUAGE plperl;
```

Dieses Beispiel[11] versucht eine Datei auf dem Server zu erstellen. Dies ist der Sprache PL/Perl aber nicht gestattet.
Schon bei der Erzeugung dieser Funktion würde der Compiler sich beschweren und das Erzeugen verweigern, da der über *open* erzeugte File-Handle die Sicherheitsbestimmungen der Sprache verletzt.
Um diese Funktion erzeugen zu können muss PL/PerlU genutzt werden. Wie schon angesprochen muss hierbei allerdings der Datenbank-Administrator für die Sicherheit sorgen. Er muss sicherstellen, dass die Funktion z.B. nicht zu anderen, als den gewollten Zwecken genutzt werden kann.

[11] vgl. http://www.postgresql.org/docs/8.1/static/plperl-trusted.html

4 google.pl - Das Perl-Skript

In dem von Google gelieferten Paket befindet sich eine wsdl-Datei, also eine Web Service Description Language Datei. Hierbei handelt es sich um eine spezielle xml-Datei, die alle benötigten Informationen für die SOAP-Schnittstelle enthält. Mittels dieser wsdl-Datei und dem von Google generierten Key kann nun eine Anfrage an www.google.com gestellt werden und die Ergebnisse werden in der Shell in Plaintext angezeigt.

```
#!perl

# google development key
my $google_key=<GoogleKey>;

# location of googleSearch.wsdl
my $google_wsdl = "./GoogleSearch.wsdl";

use strict;

# bind SOAP::Lite-module
use SOAP::Lite;

# extract query from the shell
my $query = shift @ARGV or die "Usage: perl google.pl \"<query>\"\n";

# create SOAP::Lite-instance by using $google_wsdl
my $google_search = SOAP::Lite->service("file:$google_wsdl");

print "Asking for \"".$query."\"...\n\n";

# do google search for $query
my $results = $google_search -> doGoogleSearch($google_key, $query,
                    0,10,"false", "","false","", "latin1", "latin1");

# any results?
@{$results->{resultElements}} or exit;
if ($results > 1){
  print "here comes the result (0..10)\n";
}

# iterate through the results
foreach my $result (@{$results->{resultElements}}) {
# printing the main informations
print
  join "\n",
  $result->{title} || "no title",
  $result->{URL} || "no URL",
  $result->{snippet} || "no snippet",
  "\n";
}
```

Dieses Perl-Skript[12] liefert pro Anfrage 10 Tripel beginnend mit dem ersten Treffer. Dieses Tripel besteht aus dem Titel der gefundenen Internetseite, der URL und einem Abriss über den Inhalt der Seite.
Allgemein bietet die GoogleAPI folgende Werte als Teil des Ergebnisses an:

summary Falls die Anfrage im ODP-Verzeichnis von Google gefunden wird, wird hier die Zusammenfassung des entsprechenden ODP-Eintrags angezeigt.[13]

URL Die absolute Internet-Adresse des Suchergebnisses.

snippet Kurzer Abriss der Seite, meist um die Fundstelle.

title Titel der Seite.

cachedSize Falls die Seite in gecachter Form vorliegt, so wird die Größe geliefert.

relatedInformationPresent Bool'scher Wert, der auf ähnliche Seiten verweist.

hostName Anzeige des Hostnamen ab dem 2. Treffer bei eingeschalteter Filterung[14].

directoryCategory Falls ein ODP-Eintrag existiert, werden hier die Werte fullViewableName, sowie specialEncoding ausgegeben.

directoryTitle Falls ein ODP-Eintrag existiert, so wird hier der Titel des ODP-Eintrages angezeigt.

Auf diese Werte kann in dem o.a. Perl-Skript z.B. mittels:

```
$result->{title}
```

zugegriffen werden.
Wenn man nun mehr als die ersten 10 Treffer ausgegeben bekommen möchte, so muss man die Anfrage in einer Schleife stellen, welche den Offset der Anfrage [15] um 10 heraufsetzt. Eine direkte Anfrage über mehr als 10 Treffer wird von der API nicht unterstützt und führt zu leeren Ergebnissen.

4.1 Die eigentliche Anfrage

Die Anfrage dieses Scripts besteht aus:

```
my $google_key=<GoogleKey>;
..
my $google_wsdl = "./GoogleSearch.wsdl";
..
use SOAP::Lite;
```

[12] vgl. [oreilly]
[13] ODP: Open Directoy Project (auch DMoz genannt) ist ein Open-Content-Link-Verzeichnis. Hier werden Links und z.B. kurze Inhaltsangaben zu angemeldeten Seiten gespeichert und verwaltet.
[14] vgl. Kapitel 4.1
[15] vgl. Kapitel 4.1

```
..
my $google_search = SOAP::Lite->service("file:$google_wsdl");
..
my $results = $google_search -> doGoogleSearch($google_key, $query,
                  0,10,"false", "","false","", "latin1", "latin1");
```

Wobei die eigentliche Anfrage nur aus der letzten Zeile besteht.
In dieser Zeile wird der Variablen $results das Ergebnis der Funktion doGoogleSearch(..) des Objekts $google_search übergeben.
Die Funktion doGoogleSearch benötigt als Parameter[16]:

1. den $googleKey, den man bei der Registrierung unter google.com/apis erhalten hat.
   Dieser Key wird von Google erzeugt, um die Nutzung der API zu kontrollieren und reglementieren, wodurch ein übermäßiger Datenverkehr auf der Seite von Google vermieden werden soll.
   Ohne diesen Key ist es nicht möglich Ergebnisse von Google zu erhalten.
2. den $query, also die eigentliche relevante Anfrage an Google,
3. einen Offset (hier: 0), der das erste Ergebnis angibt, das zurückgegeben werden soll.
4. die Anzahl der Ergebnisse, die pro Anfrage zurückgeliefert werden sollen. Dieser Wert darf zwischen 0 .. 10 liegen. Liegt der Wert oberhalb der 10, so wird von Google keine bzw. eine leere Antwort auf die Anfrage zurückgeliefert.
5. die Angabe eines Filters, der sich auf die Ausgabe von Ergebnissen bezieht. Hierdurch ist es möglich, dass duplizierte Ergebnisse, also Ergebnisse, die den selben Titel und den selben snippet enthalten ausgefiltert werden.
6. ein restrict, der bewirkt, dass nur Internetseiten in speziellen Ländern in die Anfrage mit einbezogen werden, z.B. bewirkt die Angabe von countryDE das Durchsuchen von deutschen Servern.
7. die Angabe über den Status SafeSearch, welcher z.B. pornografische Inhalte und Ergebnisse ausfiltert.
8. lr, ein language restrict, über den die Sprache der Ergebnisse festgelegt werden kann.
   Z.B. sind hier lang_de für deutsche, lang_eng für englische oder lang_ko für koreanische Ergebnisse möglich.
9. Die Eingabe-Codierung, sowie
10. die Ausgabe-Codierung.

[16] Für eine komplette Übersicht vgl. [googleAPI]

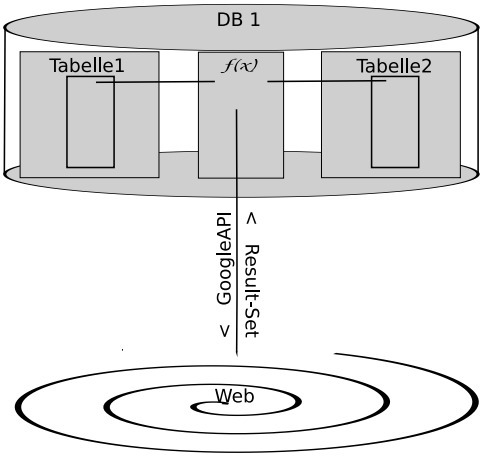

Abbildung 1: Grundarchitektur der Table Function googleJoin(..)

5 Die Table Function googleJoin(..)

5.1 Der Grundgedanke der Funktion

Die Table Function googleJoin(..) soll Spalten aus maximal 2 Tabellen einer Datenbank auslesen, diese Wertepaare zu einem String zusammenfügen und diesen über die GoogleAPI im Internet auf die Trefferhäufigkeit hin untersuchen[17].
Anstelle von 2 Tabellenspalten kann auch ein fixer String mit den Werten einer Spalte zusammengefügt werden.
Anhand einer, vorher der Funktion übergebenen, Grenze soll nun entschieden werden, ob dieser String *Sinn* macht, also mehr Treffer gefunden werden, als mit diesem Grenzwert festgelegt worden sind. Ist dies der Fall wird 'true', andernfalls wird 'false' von der Funktion geliefert.
Um den Suchraum noch genauer zu spezifizieren, soll es möglich sein der Funktion noch ein Site-Attribut mitzugeben, also ein Attribut, dass die Google-Anfrage nur auf einen speziellen Server reduziert.

Aus diesen Anforderungen ergibt sich die Signatur

```
bool googleJoin(string, string, string, int) .
```

Um nun ggf. noch weitere Informationen über den Funktionsverlauf zu erhalten wurde noch ein fünftes Attribut eingefügt, dass als Schalter für die *verbose*-Informationen fungieren soll. Dieser Schalter wird über den letzten Integer-Wert der Funktion gesteuert.
Bei der Eingabe einer '0' passiert alles leise im Hintergrund, bei einer '1' werden u.a. die eigentliche Anfrage, Statusausgaben, die gefundenen Treffer und die Entscheidung der Funktion auf der Konsole ausgegeben.

[17] vgl. Abbildung 1

5.2 Die Implementierung

Das Perl-Skript kann nahezu ohne weiteres in die Datenbank übernommen werden. Allerdings müssen bei der Portierung von Perl nach PL/PerlU einige Kleinigkeiten beachtet werden:

- Wie schon angesprochen darf nicht die normale Perl-Adaption von postgre genutzt werden. Es ist PL/PerlU zu nutzen.
- In PL/PerlU-Skripten, die eine Verbindung zu externen Servern aufbauen darf nicht der Perl-Befehl *exit* genutzt werden, da es hier zu einem nicht-kontrollierten Verbindungsabbruch kommt, worauf postgre mit einer Fehlermeldung die komplette Anfrage abbricht und keine Ergebnisse liefert.
- Ebenfalls ist die Verwendung des *print*-Befehls nicht möglich. Zur Ausgabe von Informationen ist der *elog*-Befehl notwendig. Dieser Befehl erlaubt die Ausgabe von Log-Informationen, welche je nach Bedeutung von *DEBUG, LOG*, über *INFO*, bis hinzu *WARNING, ERROR* gestaffelt sind und die, je nach Stufe, auch im eigentlichen Datenbank-Log mitprotokolliert werden.
  Ein *ERROR* bewirkt das sofortige Abbrechen der Funktion und der ausgeführten Transaktionen.

```
CREATE OR REPLACE FUNCTION googleJoin(text,text,text,int,int)
                                                    RETURNS bool AS'
  # reading given arguments
  my $str1 = $_[0];
  my $str2  = $_[1];
  my $site = $_[2];

  # threshold of hits to return a positive function result
  my $threshold = $_[3];

  # adding some output informations?
  my $verbose = $_[4];

  # number of found hits
  my $resCount = 0;

  # loop-breaking-condition
  my $resCountTemp = -1;

  # counting-variable for the offset
  my $from = 0;

  #constructing the query
  my $query = qq($str1). " " . qq($str2);
  if (!($site eq "")){
    $query = $query." site:" . $site;
  }
```

```
if ($verbose){
  elog NOTICE, "-== Query: ".$query." ==-";
}

# google development key
my $google_key=<GoogleKey>;

# location of the googleSearch.wsdl on google.com
my $google_wsdl = "http://api.google.com/GoogleSearch.wsdl";

# bind SOAP::Lite-module
use SOAP::Lite;

# create SOAP::Lite-instance by using the $google_wsdl
my $google_search = SOAP::Lite->service("$google_wsdl");

if ($verbose){
  elog NOTICE, "reading wsdl-file and creating SOAP::Lite done";
}

# while not enough hits are found
# and still finding some new hits
# ask for 10 new results resuming from the latest offset + 10
while (($from < $threshold) && ($resCountTemp!=0)){
  $resCountTemp=0;

  # for verbose informations
  my $to = $from + 10;
  my ($results) = $google_search -> doGoogleSearch($google_key,
                    $query, $from, 10,"false", "", "false","",
                                                "utf-8", "utf-8");
  if ($verbose){
    elog NOTICE, "query... done";
  }

  # number of current hits
  $resCountTemp = @{$results->{resultElements}};
  if ($verbose){
    elog NOTICE, $resCountTemp . " hits found";
  }

  # updating number of hits
  $resCount += $resCountTemp;
  if($verbose){
    elog NOTICE, "  statistics:";
    elog NOTICE, "  # from ".$from." to ".$to."! threshold: "
                    .!threshold."! overall hits: ".$resCount;
    elog NOTICE, "  ".$threshold - $resCount." hits remaining";
  }
```

```
    # updating offset
    $from +=10;
  }
  if ($verbose){
    if ($resCount >= $threshold){
      elog NOTICE, $query." OK ..";
    }else{
      elog NOTICE, $query." failed..";
    }
  }

  # result of the function
  if ($resCount >= $threshold){
    return "true";
  }else{
    return "false";
  }
' LANGUAGE plperlu;

SELECT googleJoin('NAME','FACHBEREICH','SITE',GRENZE,VERBOSE);
```

Diese Table Function liefert je nach Ergebnis eine bool'sche Antwort für die Anfrage zurück. Die Funktion liest beim Aufruf die übergebenen Attribute aus und setzt diese zu einer Anfrage zusammen.
Die ersten drei Attribute sind für die eigentliche Suchanfrage, das vierte gibt die Mindestanzahl der Treffer für ein positives Ergebnis und das letzte gibt an, ob zusätzliche Informationen ausgegeben werden sollen[18].
Hierbei werden die ersten beiden benötigt, um die beiden Tabellen miteinander in Verbindung zu setzen, das dritte kann den Suchraum der Anfragen auf einzelne Internetseiten begrenzen. Zum Beispiel kann die Suche nur auf die Seite 'www.uni-koblenz.de' begrenzt werden, wodurch irrelevante Ergebnisse von vorne herein verhindert werden können.
Für ein verlässliches Ergebnis der Beziehung zwischen einem Mitarbeiter und einem Fachbereich bzw. einer Firma oder Niederlassung ist das richtige Einstellen des site-Attributes unabdingbar. Wenn man nun z.B. nach einem Herrn 'Schmidt' und 'Informatik' global sucht, erhält man unweigerlich eine wesentlich höhere Treffermenge, als bei der Reduzierung z.B. auf den Bereich der Universität Koblenz mittels Angabe von 'www.uni-koblenz.de'. Bei der größeren Treffermenge ist nicht das Problem, dass mehr Ergebnisse geliefert werden, was bei spezifischeren Ergebnissen bei der selben Anfrage ja kein Hindernis wäre, sondern das auch Ergebnisse mit einfließen, die nicht in unseren Suchraum passen. Um bei dem Beispiel zu bleiben, sind uns ja die Ergebnisse eines Herrn 'Schmidt', der in der Schweiz oder Österreich 'Informatik' mal in der Schule gelernt hat, egal, wenn wir wissen wollen, ob Herr 'Schmidt' 'Informatik' an der Universität Koblenz hält.

[18] vgl. Kapitel 5.1

5.3 Die Tabellen employee und fb

Um nun die Table Function googleJoin(text, text, text, int, int) zu evaluieren, werden zu Testzwecken folgende zwei Tabellen erstellt:

```
CREATE TABLE employee (
    name text
);

CREATE TABLE fb (
fb_name text
);
```

In die Tabelle employee werden nun einige Testangestellte eingetragen:

```
INSERT INTO employee (name) VALUES ('Schmidt');
INSERT INTO employee (name) VALUES ('Heinrich');
INSERT INTO employee (name) VALUES ('Lüddenscheidt');
INSERT INTO employee (name) VALUES ('Steffen Staab');
INSERT INTO employee (name) VALUES ('Simon Schenk');
```

In die Tabelle fb werden folgende Bereiche erzeugt:

```
INSERT INTO fb (fb_name) VALUES ('Informatik');
INSERT INTO fb (fb_name) VALUES ('Mathe');
INSERT INTO fb (fb_name) VALUES ('Physik');
INSERT INTO fb (fb_name) VALUES ('Pädagogik');
```

Im Folgenden werden nun anhand dieser Tabellen Beispiel-Anfragen und deren Ergebnisse aufgezeigt.

5.4 Beispiel-Anfragen und Ergebnisse

Die Table Function ermöglicht es theoretisch beliebige Anfragen an Google zu stellen.

```
SELECT googleJoin('Thorsten Tillack','Informatik','www.uni-koblenz.de',
                                                              10,1);
```

erzeugt also eine Anfrage nach 'Thorsten Tillack Informatik site:www.uni-koblenz.de' und benötigt für eine positive Antwort 10 Treffer und liefert zusätzliche Informationen. Diese Anfrage erfolgt komplett unabhängig von jeglicher Tabelle. Die Datenbank wird jedoch benötigt, da die Anfrage von der Datenbank ausgehend an Google gestellt wird und die Datenbank auch das Ergebnis geliefert bekommt. Da das *verbose*-Attribut gesetzt worden ist, liefert die *SELECT*-Anfrage

```
NOTICE:  Query: Thorsten Tillack Informatik site:www.uni-koblenz.de
NOTICE:  reading wsdl-file and creating SOAP::Lite done
NOTICE:  query... done
NOTICE:  10 hits found
NOTICE:    statistics:
NOTICE:    # from 0 to 10! threshold: 10! overall hits: 10
NOTICE:  0 hits remaining...
```

```
NOTICE:  Thorsten Tillack Informatik site:www.uni-koblenz.de OK ..
 googlejoin
------------
 t
(1 Zeile)
```

Da die Anfrage recht viele Parameter hat, wurden zwei Alternativmethoden erstellt.

```
CREATE OR REPLACE FUNCTION hasGoogleJoin(text,text,int)
                                              RETURNS bool AS '
  SELECT * FROM googleJoin($1,$2,\'\',$3,0);
' LANGUAGE SQL;

CREATE OR REPLACE FUNCTION hasGoogleJoin(text,text,text,int)
                                              RETURNS bool AS '
  SELECT * FROM googleJoin($1,$2,$3,$4,0);
' LANGUAGE SQL;
```

Diese Table Functions ermöglichen ein Ausführung der Funktion googleJoin(text, text, text, int, int) mittels

hasGoogleJoin(text,text,int) , wodurch die Funktion googleJoin ohne *site*- und ohne *verbose*-Attribut (*verbose* = 0) ausgeführt wird.

hasGoogleJoin(text,text,text,int) , wodurch googleJoin ohne *verbose*-, jedoch mit *site*-Parameter gestartet wird.

5.5 Weitere Anfrage-Beispiele

Um nun den eigentlichen Sinn der Table Function zu nutzen, muss in den Funktionsaufruf an die Stelle der ersten beiden Parameter der Name der zu benutzenden Spalten eingetragen werden.
In unserer kleine Welt von *fb* und *employee* bedeutet das also:

```
SELECT fb.fb_name,employee.name,hasGoogleJoin(fb.fb_name,
                          employee.name,'www.uni-koblenz.de',10);
  fb_name   |     name      | hasgooglejoin
------------+---------------+---------------
 Informatik | Schmidt       | t
 Informatik | Heinrich      | t
 Informatik | Lüddenscheidt | f
 Informatik | Steffen Staab | t
 Informatik | Simon Schenk  | t
 Mathe      | Schmidt       | t
 Mathe      | Heinrich      | t
 Mathe      | Lüddenscheidt | f
 Mathe      | Steffen Staab | f
 Mathe      | Simon Schenk  | f
 Physik     | Schmidt       | t
 Physik     | Heinrich      | t
 Physik     | Lüddenscheidt | f
```

```
 Physik     | Steffen Staab | f
 Physik     | Simon Schenk  | f
 Pädagogik  | Schmidt       | t
 Pädagogik  | Heinrich      | t
 Pädagogik  | Lüddenscheidt | f
 Pädagogik  | Steffen Staab | f
 Pädagogik  | Simon Schenk  | f
(20 Zeilen)
```

Bei dieser Anfrage ist allerdings keine lokale Restriktion auf nur eine Internetseite erfolgt, was sinngemäß durch den Aufruf

```
SELECT fb.fb_name,employee.name,hasGoogleJoin(fb.fb_name,
                     employee.name,'www.uni-koblenz.de',10);
```

ausgeführt wird.

```
  fb_name   |     name      | googlejoin
------------+---------------+------------
 Informatik | Schmidt       | t
 Informatik | Heinrich      | t
 Informatik | Lüddenscheidt | f
 Informatik | Steffen Staab | t
 Informatik | Simon Schenk  | t
 Mathe      | Schmidt       | t
 Mathe      | Heinrich      | t
 Mathe      | Lüddenscheidt | f
 Mathe      | Steffen Staab | f
 Mathe      | Simon Schenk  | f
 Physik     | Schmidt       | t
 Physik     | Heinrich      | t
 Physik     | Lüddenscheidt | f
 Physik     | Steffen Staab | f
 Physik     | Simon Schenk  | f
 Pädagogik  | Schmidt       | t
 Pädagogik  | Heinrich      | t
 Pädagogik  | Lüddenscheidt | f
 Pädagogik  | Steffen Staab | f
 Pädagogik  | Simon Schenk  | f
(20 Zeilen)
```

Anhand diese Beispiels wird die Bedeutung der lokalen Restriktion der Anfrage deutlich.
Der Unterschied wird bei dem Tupel 'Steffen Staab' und 'Mathe' deutlich. Ohne die Begrenzung wird ein 'true' geliefert. Diese Ergebnis wird allerdings bei der Reduzierung auf 'www.uni-koblenz.de' wieder negiert. Dieses *Problem* würde bei einer höheren Grenze noch weitaus verstärkt.
Dieser Effekt zeigt, dass es sinnvoll ist die Limitation des Anfrageraumes zu nutzen.

6 Wieso keine fixe Grenze?

Bei den meisten Anwendungen ist eine fixe Grenze, die über ein positives oder negatives Ergebnis entscheidet äußerst sinnvoll. Diese fixe Grenze wird meist über einen langen Evaluationsvorgang ermittelt oder ist durch Messungen bekannt.
Ein fixer Grenzwert hat sich aber beim googleJoin als nicht sinnvoll erwiesen, da zu viele verschiedenen Faktoren in das Ergebnis der Funktion einfließen.
Einige der wichtigsten Einflußgrößen sind:

die Größe des Unternehmens Je nach Größe des Unternehmens bzw. Dauer der Betriebsexistenz sind im Internet eine unterschiedliche Anzahl an Dokumenten oder Verweisen auf das Unternehmen, die Abteilungen und deren Mitarbeitern vorhanden. Bei einem großen, etablierten Unternehmen ist ein Treffer wahrscheinlicher als bei einer neuen kleinen Firma.

die Namenshäufigkeit Abhängig vom nachgefragten Namen gibts es auch hier große Unterschiede. Für den Namen 'Schmidt' wird es mehr Ergebnisse geben als für den Namen 'Lüddenscheidt'.

den Inhalt der Spalten Ähnliches wie bei der Namenshäufigkeit gilt auch für den Inhalt der einzelnen Spalten. Wenn die Spalte der Mitarbeiter nur den Nachnamen enthält, entfallen auf diese Anfrage wahrscheinlich mehr Treffer, als bei einer stärker spezifizierten Spalte, die Vor- und Nachnamen enthält.

Verwendung von Namen Der Effekt des Spalteninhalts lässt sich je nach Name noch verstärken. Z.B. kann der Name 'Heinrich' als Vor- und Nachname auftreten, was die Suche nach dem reinen Nachnamen verfälscht. Der Name 'Lüddenscheidt' ist als Vorname jedoch eher unwahrscheinlich.

globale oder lokale Anfrage Wenn die Anfrage auf eine einzige Internetseite beschränkt wird, so wird die Trefferhäufigkeit wesentlich geringer ausfallen, als bei einer globalen Suche über Google.

Aus den obigen Faktoren lässt sich ein weiterer Grund für die fehlende Existenz einer fixen Grenze erkennen.
Je nach Existenzdauer des Suchraumes nimmt die Anzahl der Dokumente und somit der Treffer zu. Dies liegt daran, dass es sich beim Internet um einen dynamischen Bereich handelt. Täglich werden Internetseiten modifiziert; es werden Tausende, wenn nicht Millionen Seiten gelöscht und mindestens genauso viele neue Seiten erstellt. Hierdurch wird das Internet und somit auch unsere *Wissensdatenbank* modifiziert.
Durch diese Modifikation werden Grenzen, die heute noch einwandfrei funktionieren, vielleicht in 2 Wochen schon nicht mehr effektiv arbeiten und schlechte Ergebnisse liefern.

Es lässt sich also nicht einfach ermitteln, welche fixe Grenze man benötigt, sondern man muss anhand dieser Faktoren nun abschätzen, welche Grenze man der Funktion übergibt.

7 Bewertung der Funktion googleJoin(..)

Nach dem die technische Umsetzung funktioniert hat, bleibt die Frage, wie effektiv bzw. brauchbar dieses Verfahren ist.
Die Effektivität des eigentlichen Suchens muss nicht in Frage gestellt werden, da die GoogleAPI direkt auf das Suchverfahren von www.google.com zugreift. Bleibt also das eigentliche Verfahren des googleJoins zu überprüfen.
Hierzu wird die Mitarbeiterdatenbank der Universität Koblenz mit den Ergebnissen der Anfrage von Hand verglichen.

- 'Informatik Schmidt' liefert mittels der Funktion ein 'true', was über die Mitarbeiter-Seite des Fachbereiches 4 nur bestätigt werden kann.
- 'Informatik Heinrich' wird auch durch die Funktion und die Mitarbeiterseite bestätigt.
- 'Informatik Lüddenscheidt' wird mit einem 'false' bewertet, was durch die Recherche auch verifiziert wird.
- Ähnlich verhält es sich für die positiven Ergebnisse zu 'Informatik Steffen Staab' und 'Informatik Simon Schenk'.[19]

Auch die anderen Ergebnisse der Funktion werden durch die *manuelle* Kontrolle auf den Mitarbeiterseiten bestätigt.
Daraus folgt, dass googleJoin brauchbare Ergebnisse mit vergleichsweise wenig Aufwand liefert. Der Aufwand bezieht sich hierbei auf das Stellen der Anfrage; bei einem neuen Namen, den man überprüfen will, müsste man m neue Anfragen stellen, wobei m die Anzahl der 'Fachbereiche' ist.
Bei der Nutzung der Table Function muss man nur den Wert in die entsprechende Tabelle eintragen und mittels

- `SELECT hasGoogleJoin('NEUER_NAME',fb.fb_name,GRENZE);`

  für die fehlenden Ergebnisse zu dem neuen Namen oder

- `SELECT hasGoogleJoin(employee.name,fb.fb_name,GRENZE);`

  für eine komplett neue Erstellung der Ergebnisspalte

eine neue Anfrage stellen. Beide Male ist der Aufwand geringer als bei dem Stellen manueller Anfragen an Google.

[19] vgl. www.uni-koblenz.de/FB4/People

8 Verbesserung von googleJoin

Nachdem die *Vertrauenswürdigkeit* der Funktion bzw. der Effektivität nun bewiesen ist, bleibt zu überlegen, mit welchen Mitteln der GoogleAPI sich die Effektivität und die Zuverlässigkeit des googleJoin noch steigern lassen.
Wie schon in Kapitel 4 aufgezeigt, gibt es verschiedene Einstellungsmöglichkeiten für das zurückgelieferte Ergebnis der Anfrage. Aber welche dieser zusätzlichen Ausgabemöglichkeiten erhöht die Trefferqualität?
Die für diesen Zweck relevanten Ausgaben der API-Anfrage wären:

- URL
- summary
- snippet
- title

Bei den übrigen Ausgaben wurden entweder zu spezifische Ergebnisse geliefert oder es wurde lediglich ein Hinweis auf andere (ähnliche) Seiten geliefert, was für die Trefferauswertung nicht von Bedeutung war.

8.1 Hierarchisierung der Treffer

Nun gilt es eine erste Hierarchie zwischen den einzelnen Treffern zu erzeugen. Hierbei könnte man anhand der Trefferwahrscheinlichkeit vorgehen. Was zu der Frage führt:
Wie wahrscheinlich ist es, dass die gesuchte Anfrage in den Treffer vorkommt?
Das Maß der Trefferwahrscheinlichkeit hängt stark von der Länge des Ergebnisses ab. Je nach gewünschter Antwort (also: URL, summary, snippet oder title) variert diese Länge jedoch erheblich.
Falls vorhanden wird die summary den größten Treffer darstellen, gefolgt von dem snippet, dem title und der URL.
Bei einer hohen Trefferwahrscheinlichkeit ist jedoch der Fund der Anfrage nicht so schwer zu gewichten, wie bei einer geringeren Trefferchance. Also würde ein Treffer in der URL eine höhere Bedeutung erhalten, als ein Treffer in der summary.

Eine andere Herangehensweise wäre, den Kontext der Seite mit in die Bewertung einfließen zu lassen.
Der Kontext als Einflussgröße benötigt aber eine andere Trefferbewertung als das reine Finden eines Treffers. Wenn im ODP eine summary der Seite angegeben ist und in dieser Inhaltsangabe ein Treffer gefunden wird, so kann davon ausgegangen werden, dass dieser Treffer *gewollt* ist und somit eine wesentlich höher Bedeutung zu tragen hat, als oben angenommen. Im Rang der Bedeutung des Treffers würde nun vermutlich ein Treffer im title der Seite kommen, gefolgt vom snippet und der URL.
Allerdings ist zu beachten, dass die URL heutzutage meist nur noch aus dem *domain name* besteht, was einen Treffer mittels der GoogleAPI recht unwahrscheinlich macht. Demzufolge müsste also ein Treffer in der URL eine wesentlich

stärkere Bedeutung erhalten; stärker als die Bedeutung eines Treffers im title, da sich der Titel einer Seite auch mehrmals (je nach angezeigtem Inhalt) ändern kann.
Um beim Kontext-Ansatz zu bleiben folgt somit diese Reihenfolge der Bewertung eines Treffers:

1. summary (falls vorhanden)
2. URL
3. title
4. snippet

Mit dieser Reihenfolge müsste sich nun die Berechnung des *$resCount*'s[20] leicht modifizieren lassen.

8.2 Eine erste Annäherung

Angenommen ein Treffer in der summary würde den $resCount direkt um einen Wert von 10 erhöhen. Hierdurch würde vermutlich die Grenze ($threshold) bei kleinen Anfrage-Zielgebieten schnell oder sogar direkt gesprengt. Also muss hier eine andere Art der Bewertung einfließen.

- Die Bewertung der Kontext-Treffer wird in einer Variablen $cRes gespeichert.
- Für Treffer in der summary werden 10, für die URL 5, für den title 2, und den snippet 1 'Punkt' addiert.
- Ziel der kontextgetriebenen Bewertung soll es vorerst sein, dass fälschlicher Weise 'false' bewertete Ergebnisse durch den Einfluss von $cRes ggf. korrigiert werden.
- Hierfür muss die Bedeutung eines Kontext-Treffers im Vergleich zu einem *normalen* Treffer betrachtet werden.
  Da ein Kontext-Treffer höher bewertet werden soll, als ein *normaler* Treffer folgt: $\$resCount < \$cRes$
- Ein *normaler* Treffer wird als 'true' gesehen, falls:
  $1 \leq \frac{\$resCount}{\$threshold} < \frac{\$cRes}{\$threshold}$
- Hieraus folgt:
  $1 \geq \frac{\$threshold}{\$resCount} > \frac{\$threshold}{\$cRes}$
- Das bedeutet, wir erhalten als Bedingung für eine positive Rückmeldung der Funktion:
  $\$cRes \geq \frac{\$threshold * \$cRes}{\$resCount} > \$threshold$

[20] vgl. Kapitel 5.2

Da es nun aber Seiten geben wird, bei denen es keinerlei Treffer in der Kontext-Abteilung der Bewertung geben wird, muss die alte Art der Entscheidungsfindung in der Funktion enthalten bleiben. Dies ergibt sich auch aus der Überlegung, dass Kontext-Treffer im ersten Schritt nur unterstützend zu den *normalen* Treffern zu sehen sind und 'false' deklarierte Wertepaare ggf. auf 'true' gesetzen werden sollen.
Mit alldiesen Überlegungen kann nun die Bewertungsmethodik der Table Function modifiziert werden.

```
# PSEUDO-CODE
Solange noch Treffer zu finden sind und noch welche gefunden
                                                bzw. benötigt werden{
  Ergebnisse = googleAPI->Anfrage;
  Anzahl der Treffer = Dimension(Ergebnisse);

  #Über Ergebnisse iterieren
  Für jedes Ergebnis{
    Falls Ergebnis.summary Anfrage enthält: KontextCount += 10;
    Falls Ergebnis.URL Anfrage enthält: KontextCount += 5;
    Falls Ergebnis.title Anfrage enthält: KontextCount += 2;
    Falls Ergebnis.snippet Anfrage enthält: KontextCount += 1;
  }
}

Falls genug Ergebnisse gefunden wurden oder
  ((KontextCount * Grenzwert) / Anzahl der Ergebnisse) > Grenze{
  return 'true';
}else{
  return 'false';
}
```

8.3 C-PANKOW meets googleJoin

Dieses Verfahren spiegelt einen Teil von C-PANKOW [21] wieder. Es bezieht den Kontext der Trefferumgebung mit ein um bessere bzw. genauere Ergebnisse zu liefern. Allerdings ist es nicht so effizienzsteigernd wie das *normale* C-PANKOW im Vergleich zum *normalen* PANKOW.
Bei C-PANKOW werden die Inhaltsangaben heruntergeladen und offline verarbeitet, wodurch eine geringere Arbeitsdauer entsteht und direkt irrelevante Anfrage ausgefiltert werden können. Dies geschieht bei googleJoin nicht. Es werden die Informationen direkt durch die GoogleAPI bezogen und es wird verglichen, ob die beiden Teile der Suchanfrage in den einzelnen Bestandteilen des Ergebnisses enthalten sind. Im Vergleich zu C-PANKOW ist dies zwar keine Reduzierung des Datenaufkommens, allerdings werden diese Ergebnismengen [22] der Anfrage jedesmal heruntergeladen, egal ob man sie auswertet oder nur zählt. Also entsteht im Vergleich zu googleJoin kein größerer Datenverkehr, es erhöht sich lediglich minimal die eigentlich Rechenzeit pro Anfrage. Diese Vermehrung von *Wartezeit* ist allerdings im Bezug auf die Dauer, die benötigt wird

[21] vgl. Kapitel 2.2
[22] Ergebnismenge als Tupel aus: summary, title, URL ... pro Treffer

um die wsdl-Datei von www.google.com zu beziehen, zu gering um großartig ins Gewicht zu fallen.
Um nun die bestehende Table Function um die Kontext-Bewertung zu erweitern sind noch ein paar zusätzliche Variablen notwendig:

$cRes Speichert die Bewertung der Kontext-Treffer

$count Speichert die Anzahl der Schleifendurchgänge, da bei der Durchsuchung einer speziellen Internetseite diese nur einmal bewertet werden darf.

```
# [..]
my $cRes = 0;
# [..]

while (($from < $threshold) && ($resCountTemp!=0)
       && ((($threshold*$cRes)/($resCount+1))<=(2*$threshold))){
  $resCountTemp=0;

  # for verbose informations
  my $to = $from + 10;
  my ($results) = $google_search -> doGoogleSearch($google_key,
                    $query, $from, 10,"false", "", "false","",
                                "utf-8", "utf-8");
  # number of current hits
  $resCountTemp = @{$results->{resultElements}};
  if ($verbose){
    # [..]
  }

  # updating number of hits
  $resCount += $resCountTemp;
  if($verbose){
    # [..]
  }

  my $count = 0;
  foreach my $result (@{$results->{resultElements}}) {
    $count++;
    # loop-breaker
    if($site eq "") {
      if($count == 2){
        last;
      }
    }
    # result elements contain query parts?
    if((rindex $result->{summary},$str1 != -1)
                && (rindex $result->{summary},$str2 != -1)){
      $cRes += 10; next;
    }
    if((rindex $result->{URL},$str1 != -1)
                && (rindex $result->{URL},$str2 != -1)){
```

```
      $cRes += 5; next;
    }
    if((rindex $result->{title},$str1 != -1)
                && (rindex $result->{title},$str2 != -1)){
      $cRes += 2; next;
    }
    if((rindex $result->{snippet},$str1 != -1)
                && (rindex $result->{snippet},$str2 != -1)){
      $cRes += 1; next;
    }
  }

  # updating offset
  $from +=10;
}
# result of the function
if (($resCount >= $threshold) ||
    ((($threshold*$cRes)/($resCount+1)) > 2 * $threshold )){
  return "true";
}else{
  return "false";
}
# [..]
```

Durch diese Modifikation der bestehenden Table Function[23] werden nicht nur die reinen Treffer gewertet, sondern es wird auch berücksichtigt, wo die Treffer erfolgen.
Je nach Art des Treffers wird dieser auch unterschiedlich gewichtet [24]. Diese Gewichtung ist, je nach Art der Anfrage, verschieden zu behandeln.
Beschränkt sich die Anfrage nur auf eine einzelnen Seite (ist also das $site-Attribut gesetzt), so werden die Kontext-Ergebnisse nur einmal in $cRes einbezogen. Ist die Anfrage allerdings nicht lokal beschränkt, so *muss* jede Kontextinformation neu bewertet werden, da sich die Treffer ja über verschiedene Seiten erstrecken.
Im Vergleich zu Kapitel 8.2 ist die zweite Bedingung für eine positive Antwort der Funktion leicht modifiziert. Die Untergrenze von $\frac{\$threshold*\$cRes}{\$resCount}$ ist $2 * \$threshold$. Dieser kleine Unterschied hat sich in der Testphase der Implementierung als *besser* herausgestellt, da so die Treffer gezielter als 'true' markiert werden und somit die Filterung der sinnvollen von den nicht sinnvollen Ergebnissen genauer ist.
Außerdem wurde der Nenner von $\$resCount$ aus $\$resCount + 1$ erhöht. Dies hat einen rein prakmatischen Grund:
Auf diese Weise wird eine Division durch Null verhindert, falls einmal keine Treffer für die Anfrage gefunden werden. Der Einfluss auf das Funktionsergebnis ist minimal, da diese Bedingung nur dann berücksichtigt wird, wenn die *normale* Funktion schon ein 'false' geliefert hätte. Wenn nun also der Nenner minimal vergrößert wird, kann es maximal passieren, dass sich das Ergebnis nicht ändert und *fälschlicher* Weise auf 'false' bleibt, was die Funktion ja eh geliefert hätte.

[23] vgl. Kapitel 5.2
[24] vgl Kapitel 8.1

9 Fazit

Es wurde gezeigt, dass Datenbank zur Verifikation von Daten oder Zusammenhängen durchaus das Internet als Referenz nutzen können, wobei dies mit einem recht geringen Aufwand zu bewerkstelligen ist.
Es ist lediglich eine leistungsstarke Datenbank notwendig, die eine prozedurale Sprache unterstützt. Diese Sprache muss in der Lage sein, eine Verbindung zum Internet herzustellen und die Ergebnisse in einer gewünschten Weise zu verarbeiten[25].
In unserem Fall handelt es sich um eine postgre-Datenbank, welche PL/PerlU - einen Perl-Dialekt - unterstützt. PL/PerlU[26] ermöglicht es

1. von einem Datenbanksystem gesteuert zu werden.
2. auf (mehrere) Tabellen der Datenbank zuzugreifen.
3. Tabellen zu joinen und diesen Join auszugeben oder
4. diesen Join zeilenweise weiterzuverarbeiten um
5. diese Zeilen über das Internet in eine Ontologie einzuordnen.
   (hier: Angestellte in ihren Fachbereich);
6. das Ergebnis (mittels einer angegebenen Grenze) zu bewerten und
7. eine bool'sche Aussage über den Wahrheitsgehalt der Anfrage zu liefern.

Zur Verifikation wurde im Speziellen die GoogleAPI[27] benutzt. Diese API ermöglicht den direkten Zugriff auf die Suchmaschine www.google.com und liefert die Ergebnisse an die Funktion zurück.
Für ein besseres Handling wurde dieser gesamte Ablauf in einer Table Function realisiert. Hierdurch wird das Ergebnis wieder als Spalte einer Tabelle angezeigt und kann somit wie jede andere Tabelle der Datenbank behandelt werden[28].

9.1 Der normale googleJoin

Der normale googleJoin[29] bewertet lediglich die gefundenen Treffer und vergleicht deren Anzahl mit einer eingegebenen Grenze.
Durch den Funktionsaufruf

```
SELECT * FROM googleJoin(<TabX>.spalteX, <TabY>.spalteY,
                         <Site>| '', <Grenze>, <Verbose>);
```

[25] vgl. Kapitel 3
[26] vgl. Kapitel 3.4, 3.5, 5,4
[27] vgl. Kapitel 3.1
[28] vgl. Kapitel 3.3
[29] vgl. Kapitel 5

wird googleJoin für jedes Paar $\{(x, y) \bullet x \in |spalteX| \wedge y \in |spalteY|\}$ eine Anfrage an Google gesendet und jeweils deren Treffer gezählt.
Sollte die Anzahl der Treffer > Grenze sein, so liefert googleJoin ein 'true' in der entsprechenden Reihe.
Diese Funktion liefert, soweit wie getestet werden konnte, recht zuverlässige Werte, allerdings dauert der Funktionsablauf je nach Tabellengröße und Grenze recht lange, da immer nur 10 Ergebnisse von Google zurückgeliefert werden können.

Um den Schreibaufwand der Anfrage zu reduzieren wurde eine Funktion

```
hasGoogleJoin(<TabX>.spalteX,<TabY>.spalteY,[<Site>|''], <Grenze>);
```

implementiert. Diese Funktion verweist direkt auf den googleJoin, setzt allerdings die Werte für *Verbose* und ggf. *Site* auf 'false'.

9.2 Der Kontext-googleJoin

Um die Treffergenauigkeit von googleJoin zu Verbesserung wurde eine Table Function googleJoinC[30] implementiert.
Diese Funktion hat die selbe Signatur und Funktionsweise wie der *normale* googleJoin, allerdings sieht hier die Bewertung der Treffer anders aus.
Durch die GoogleAPI werden verschiedene Antwortstrings im Ergebnis bereitgestellt. Diese Strings werden nun auf das Vorkommen der Anfrage hin untersucht. Je nach *Ort* des Treffers wird dieser höher oder niedriger bewertet.
So bringt ein Treffer

- in der 'summary' 10,
- in der 'URL' 5,
- im 'title' 2 und
- im 'snippet' der Seite 1 *Punkt.*

Diese Punkte werden nun addiert und sobald $\frac{<Grenze> * <Punkte>}{<Treffer\ insgesamt>} > <Grenze>$ überschritten ist, wird auf jeden Fall ein 'true' geliefert.
Diese Bedingung fließt allerdings nur dann in das Ergebnis ein, wenn die vorherige Bewertung fehlgeschlagen ist, d.h. es werden keine Treffer von 'true' auf 'false' gesetzt, sondern nur umgekehrt.
Diese Variante des googleJoins hat den Vorteil, dass die Ergebnisse

- mindestens genau so gut,
- auf den Kontext bezogen genauer,
- und genauso *datenintensiv*

[30] vgl. Kapitel 8.3

wie die Ergebnisse des Vorgängers sind.

Bei beiden Funktionen ist allerdings das Einstellen der Grenze zu beachten[31]. Je nach Zielgebiet der Anfrage muss eine unterschiedliche Grenze ermittelt werden. Diese Grenze ist im Laufe der Zeit höchstwahrscheinlich auch nachträglich noch zu modifizieren, da es sich beim Internet um einen dynamischen Raum handelt, welcher sich sekündlich ändert.
Deshalb ist es nicht möglich bzw. sinnvoll eine fixe Grenze in die Funktion einzubauen.
Ohne diese Modifikation der Grenze ist das Ergebnis der Table Functions mit hoher Vorsicht zu behandeln.

Für unser Szenario ist eine Grenze zwischen 10 und 15 bei der Reduzierung des Suchbereiches auf die Seite 'www.uni-koblenz.de' bei ersten Tests schon recht vielversprechend gewesen.
Eine genauere Evaluation der Grenzen im Zusammenhang mit dem Suchraum war aber leider zeitlich nicht mehr möglich.
Desweiteren war es auch nicht mehr möglich das Bewertungsschema von googleJoinC [32] genauer zu untersuchen. Es bleibt also offen, ob die Bewertung der einzelnen Treffer in diesem Rahmen in das Ergebnis einfließen müssen oder die Gewichtung korrigiert, wenn nicht sogar die Hierarchie der Treffer untereinander verändert werden muss.

Abschließend kann also gesagt werden, dass es durchaus möglich ist das Internet als Datenbank zu sehen und die Inhalte zur Verifikation von Zusammenhängen zu nutzen.

Diese Studienarbeit ist nur durch mich persönlich und - außer der angegebenen Literatur - ohne die Hilfe Dritter entstanden.

Datum T. Tillack (Matrikelnummer: 202120892)

[31] vgl. Kapitel 6
[32] vgl. Kapitel 8.2

10 Literatur

Zur Erstellung dieser Studienarbeit wurden folgende Dokumente und Internetadressen verwendet:

www04 zum Thema PANKOW:
http://www.uni-koblenz.de/ staab/Research/Publication/2004/WWW04-cimiano-handschuh-staab.pdf

www05 zum Thema C-PANKOW:
http://www.uni-koblenz.de/ staab/Research/Publication/2005/www05.pdf

googleAPI Referenz zur GoogleAPI:
http://www.google.com/apis/reference.html

postgre74 Referenz zu postgre Version 7.4.X:
http://www.postgresql.org/docs/7.4/static/index.html

postgre81 Referenz zu postgre Version 8.1:
http://www.postgresql.org/docs/8.1/static/index.html

plperl74 Referenz der prozeduralen Sprache PL/Perl(U):
http://www.postgresql.org/docs/7.4/static/plperl.html

plperl81 Referenz der prozeduralen Sprache PL/Perl(U):
http://www.postgresql.org/docs/8.1/static/plperl.html

oreilly O'Reilly Press Reading examples:
http://www.oreilly.de/catalog/googlehksger/chapter/ch05.html

sowie ggf. genauere Literaturverweise in den Fußzeilen.

11 Quelltexte

Da einige Quelltexte hier dem Layout zum Opfer gefallen sind, hier nocheinmal die kompletten Quelltexte nur minimal modifiziert.

11.1 googleJoin()

```
CREATE OR REPLACE FUNCTION googleJoin(text,text,text,int,int)
                                                RETURNS bool AS'
  # reading given arguments
  my $str1 = $_[0];
  my $str2  = $_[1];
  my $site = $_[2];

  # threshold of hits to return a positive function result
  my $threshold = $_[3];

  # adding some output informations?
  my $verbose = $_[4];

  # number of found hits
  my $resCount = 0;

  # loop-breaking-condition
  my $resCountTemp = 1;

  # counting-variable for the offset
  my $from = 0;

  #constructing the query
  my $query = qq($str1)." ".qq($str2);
  if (!($site eq "")){
    $query = $query." site:" . $site;
  }

  if ($verbose){
    elog NOTICE, "-== Query: ".$query." ==-";
  }

  # google development key
  my $google_key=<GoogleKey>;

  # location of the googleSearch.wsdl on google.com
  my $google_wsdl = "http://api.google.com/GoogleSearch.wsdl";

  # bind SOAP::Lite-module
  use SOAP::Lite;

  # create SOAP::Lite-instance by using the $google_wsdl
  my $google_search = SOAP::Lite->service("$google_wsdl");
```

```
if ($verbose){
  elog NOTICE, "reading wsdl-file and creating
                                SOAP::Lite-instance... done";
}

# while not enough hits are found
# and still finding some new hits
# ask for 10 new results resuming from the latest offset+10
while (($from < $threshold) && ($resCountTemp!=0)){
  $resCountTemp=0;

  # for verbose informations
  my $to = $from + 10;
  my ($results) = $google_search -> doGoogleSearch($google_key,
                    $query, $from, 10,"false", "", "false","",
                                              "utf-8", "utf-8");
  if ($verbose){
    elog NOTICE, "query... done";
  }

  # number of current hits
  $resCountTemp = @{$results->{resultElements}};
  if ($verbose){
    elog NOTICE, $resCountTemp . " hits found";
  }

  # updating number of hits
  $resCount += $resCountTemp;
  if($verbose){
    elog NOTICE, "  statistics:";
    elog NOTICE, "  # from ".$from." to ".$to."! threshold: "
                  .$threshold."! overall hits: ".$resCount;
    elog NOTICE, "  ".$threshold - $resCount." hits remaining";
  }

  # updating offset
  $from +=10;
}
if ($verbose){
  if ($resCount >= $threshold){
    elog NOTICE, $query." OK ..";
  }else{
    elog NOTICE, $query." failed..";
  }
}

# result of the function
if ($resCount >= $threshold){
  return "true";
```

```
  }else{
    return "false";
  }
' LANGUAGE plperlu;

CREATE OR REPLACE FUNCTION hasGoogleJoin(text,text,int)
                                        RETURNS bool AS '
  SELECT * FROM googleJoin($1,$2,\'\',$3,0);
' LANGUAGE SQL;

CREATE OR REPLACE FUNCTION hasGoogleJoin(text,text,text,int)
                                           RETURNS bool AS '
  SELECT * FROM googleJoin($1,$2,$3,$4,0);
' LANGUAGE SQL;
```

11.2 googleJoinC()

```
CREATE OR REPLACE FUNCTION googleJoinC(text,text,text,int,int)
                                                    RETURNS bool AS'
  # reading given arguments
  my $str1 = $_[0];
  my $str2  = $_[1];
  my $site = $_[2];

  # threshold of hits to return a positive function result
  my $threshold = $_[3];

  # adding some output informations?
  my $verbose = $_[4];

  # number of found hits
  my $resCount = 0;

  # loop-breaking-condition
  my $resCountTemp = 1;

  # counting-variable for the offset
  my $from = 0;

  # context-result
  my $cRes = 0;

  #constructing the query
  my $query = qq($str1)." ".qq($str2);
  if (!($site eq "")){
    $query = $query." site:" . $site;
  }

  if ($verbose){
    elog NOTICE, "-== Query: ".$query." ==-";
  }

  # google development key
  my $google_key=<GoogleKey>;

  # location of the googleSearch.wsdl on google.com
  my $google_wsdl = "http://api.google.com/GoogleSearch.wsdl";

  # bind SOAP::Lite-module
  use SOAP::Lite;

  # create SOAP::Lite-instance by using the $google_wsdl
  my $google_search = SOAP::Lite->service("$google_wsdl");

  if ($verbose){
```

```
    elog NOTICE, "reading wsdl-file and creating
                                SOAP::Lite-instance... done";
  }

  # while not enough hits are found
  # and still finding some new hits
  # ask for 10 new results resuming from the latest offset+10

  while (($from < $threshold) && ($resCountTemp!=0)){
    $resCountTemp=0;

    # for verbose informations
    my $to = $from + 10;
    my ($results) = $google_search -> doGoogleSearch($google_key,
                     $query, $from, 10, "false", "", "false","",
                                                  "utf-8", "utf-8");
    # number of current hits
    $resCountTemp = @{$results->{resultElements}};
    if ($verbose){
      elog NOTICE, $resCountTemp . " hits found";
    }

    # updating number of hits
    $resCount += $resCountTemp;
    if($verbose){
      elog NOTICE, "  statistics:";
      elog NOTICE, "  # from ".$from." to ".$to."! threshold: "
                   .$threshold."! overall hits: ".$resCount;
      elog NOTICE, "  ".$threshold - $resCount." hits remaining";
    }
    my $count = 0;
    foreach my $result (@{$results->{resultElements}}) {
      $count++;
      if($count==1 && (rindex $result->{summary},$str1 != -1)
                   && (rindex $result->{summary},$str2 != -1)){
        $cRes += 10; next;
      }
      if($count==1 && (rindex $result->{URL},$str1 != -1)
                   && (rindex $result->{URL},$str2 != -1)){
        $cRes += 5; next;
      }
      if($count==1 && (rindex $result->{title},$str1 != -1)
                   && (rindex $result->{title},$str2 != -1)){
        $cRes += 2; next;
      }
      if($count==1 && (rindex $result->{snippet},$str1 != -1)
                   && (rindex $result->{snippet},$str2 != -1)){
        $cRes += 1; next;
      }
    }
```

```
    # updating offset
    $from +=10;
  }

  # result of the function
  if (($resCount >= $threshold) ||
       ((($threshold*$cRes)/($resCount+1)) > 2 * $threshold )){
    return "true";
  }else{
    return "false";
  }
' LANGUAGE plperlu;

CREATE OR REPLACE FUNCTION hasGoogleJoin(text,text,int)
                                        RETURNS bool AS '
  SELECT * FROM googleJoinC($1,$2,\'\',$3,0);
' LANGUAGE SQL;

CREATE OR REPLACE FUNCTION hasGoogleJoin(text,text,text,int)
                                          RETURNS bool AS '
  SELECT * FROM googleJoinC($1,$2,$3,$4,0);
' LANGUAGE SQL;
```