Informe Técnico - Technical Report
DPTOIA-IT-2005-002
Enero, 2005

Un motor de búsqueda para la investigación experimental en Recuperación de la Información mediante SQL.

Carlos G. Figuerola
José Luis Alonso Berrocal
Ángel F. Zazo Rodríguez
Emilio Rodríguez Vázquez de Aldana

Departamento de Informática y Automática
Universidad de Salamanca

Revisado por

Prof. Da. Raquel Gómez Díaz
Departamento de Biblioteconomía y Documentación
Universidad de Salamanca

Prof. D. Iván Álvarez Navia
Departamento de Informática y Automática
Universidad de Salamanca

Información de los autores:

Dr. Carlos G. Figuerola
Dr. José Luis Alonso Berrocal
Dr. Ángel F. Zazo Rodríguez
Emilio Rodríguez Vázquez de Aldana

Todos ellos son integrantes del
Grupo de Recuperación Automatizada de la Información (REINA)
Área de Lenguajes y Sistemas Informáticos.
Departamento de Informática y Automática.
Facultad de Traducción y Documentación. Universidad de Salamanca
C/ Francisco Vitoria, 6-16. 37008 - Salamanca
{figue,berrocal,afzazo,aldana}@usal.es
http://reina.usal.es

Resumen

La Recuperación de Información experimenta en los últimos tiempos un auge notable, debido a la disponibilidad cada vez mayor de documentos en formato electrónico. Uno de los campos de investigación es la experimentación con diversos algoritmos referentes a cualquiera de las fases que pueden darse en el proceso de recuperación. Dicha investigación requiere, entre otras cosas, de una serie de herramientas o intrumentos que permitan la realización de experimentos. Entre esos instrumentos están los motores de recuperación; este trabajo expone cómo diseñar un motor de recuperación utilizando un Sistema de Base de Datos Relacional, y sentencias SQL.

Abstract

Research on Information Retrieval shows a remarkable growth nowadays, due to the availability of documents in electronic format. One of the research fields is the experimentation with algorithms referring to any of the tasks that can occur in the retrieval process. This research requires, among other things, of tools that allow the accomplishment of experiments. Between those tools they are the search engines; this report shows how to design a search engine using a Relational Data Base Management System and SQL sentences.

Índice

1. Introducción

La Recuperación de la Información, aunque no es precisamente un área de investigación reciente, experimenta en los últimos tiempos un auge notable, debido a la disponibilidad cada vez mayor de documentos en formato electrónico. El desarrollo y generalización del uso de Internet ha puesto de manifiesto las carencias y los retos en este campo, de manera que son numerosos los grupos de investigadores que dirigen sus esfuerzos hacia estas materias.

Uno de los campos de investigación en RI es la experimentación con diversos algoritmos, referentes a cualquiera de las fases o tareas que pueden darse en el proceso de recuperación. La investigación experimental en este campo, sin embargo, requiere, además de los conocimientos básicos necesarios, de una serie de herramientas o instrumentos que permitan la realización de experimentos.

Entre tales instrumentos, podemos distinguir, a grandes rasgos, los siguientes:

- Colecciones de documentos adecuadas, tanto por sus características documentales, como lingüísticas, e incluso de tamaño. Estas colecciones no sólo incluyen simplemente documentos, sino también baterías de preguntas o consultas, así como las correspondientes estimaciones de relevancia para las mismas [1].

- Programas que permitan indizar los documentos y resolver las consultas [18]

- Medidas eficaces y aceptadas ampliamente por la comunidad científica, que permitan evaluar y comparar los resultados de los experimentos [14, 13]

Este trabajo se centra en la producción de herramientas comprendidas en el segundo punto, esto es, de programas capaces de indizar documentos y resolver consultas. Más concretamente, en la producción de un motor experimental de recuperación, que permita utilizar alternativamente y con facilidad distintos algoritmos.

2. Motores experimentales de recuperación

Básicamente, un motor de recuperación es un programa (o un conjunto de) que es capaz de indizar documentos y de resolver o ejecutar preguntas o consultas sobre tales documentos. Sus componentes, a grandes rasgos, pueden esquematizarse de la siguiente manera:

1. Análisis léxico, es decir, la extracción de términos clave que han de representar el contenido de cada documento. Este análisis léxico puede consistir en un simple *parsing* o en procesos más complejos, como la lematización, el etiquetado semántico, etc.

2. Indización, o construcción de índices que permitan acceder a los documentos; este proceso incluye la determinación del poder descriptivo de cada uno de los términos extraídos en la fase anterior, puestos en relación con los términos de los demás documentos.

3. Resolución de consultas, o la estimación de la similitud o adecuación entre una consulta y cada uno de los documentos de la colección

4. Interfaz de usuario, que debe permitir a éste tanto formular sus necesidades informativas como obtener los resultados de las búsquedas, es decir, interactuar con el sistema. Esta interacción puede incluir elementos más complejos, como la realimentación de consultas, la selección de nuevos términos de búsqueda, la visualización de documentos o resúmenes de éstos, etc.

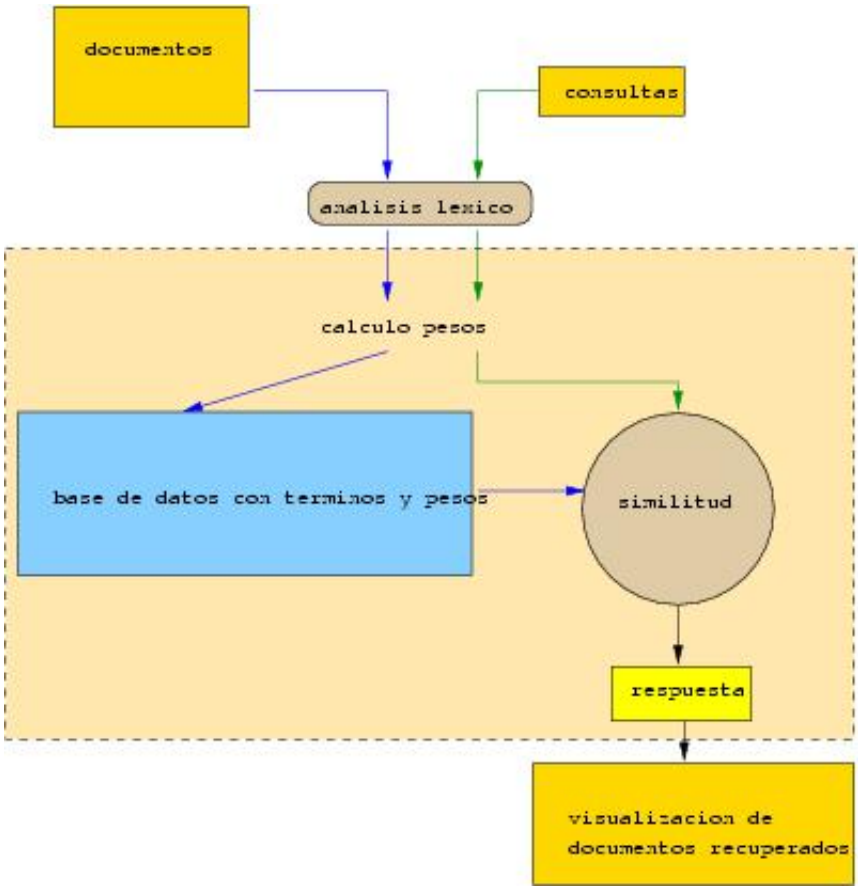

Figura 1: Esquema de un Motor de Recuperación

No obstante, suele entenderse que el corazón o núcleo, lo que realmente constituye un motor de recuperación, son los componentes 2 y 3 mencionados antes.

Existen, como es bien sabido numeroso motores de recuperación *operacionales*, diseñados para trabajar en entornos *reales*. Cada uno implementa un modelo teórico y utiliza un juego de algoritmos fijo; deben atender a las necesidades del mundo real, como, por ejemplo, la velocidad en la ejecución; y, debido a esto, además de razones comerciales en muchos casos, presentan una codificación específica destinada a resolver de la forma más eficiente posible (velocidad, consumo de recursos de máquina) sus tareas de una manera fija.

Los motores *experimentales*, sin embargo, están destinados a la experimentación y no están constreñidos por factores como la velocidad de ejecución, o al menos no lo están de forma determinante. Su misión es admitir diversas vías de resolución de problemas, en distintos entornos y con distintos objetivos específicos. A grandes rasgos, las características deseables son las siguientes:

- los componentes deben ser independientes entre sí, de manera que sea factible operar sobre parte de ellos, modificándolos, sin necesidad de tener que tocar el resto. Un motor experimental debería ser independiente de, por ejemplo, el analizador léxico, de forma que fuera posible alterar el comportamiento de éste o incluso sustituirlo por otro con diferentes capacidades
- el motor debe ser flexible como para incluir diversos algoritmos o aproximaciones a las tareas que debe resolver
- debería permitir la observación de resultados intermedios, incluso su manipulación o mo-

dificación

- el código debería ser lo más sencillo y modular posible, para facilitar su modificación
- en relación con el punto anterior, el código debería ser abierto y libremente disponible, así como estar escrito en versiones estándar de lenguajes estandar

Lamentablemente, existen pocos motores experimentales, y que cumplan las condiciones mencionadas menos. Existen motores experimentales que no son abiertos y que sólo pueden operar los investigadores que los diseñaron, y existen motores no experimentales que son utilizados -con grandes dificultades- por algunos grupos de investigación.

Uno de los paradigmas de motor experimental, utilizado durante años por diferentes grupos de investigación, es el conocido *smart* [15]. Sin embargo, *smart*, que ha prestado una ayuda inestimable a muchos investigadores [4, 5, 3], y que es una excelente herramienta de experimentación, tiene algunos inconvenientes:

- está escasamente documentado, en lo que se refiere a operación y estructura interna. Sorprendentemente, además de la magra documentación ofrecida por sus autores junto con el programa, el recurso más conocido es un breve curso de utilización básica; ni una ni otro cubren más que las capacidades más elementales del programa
- sus componentes están fuertemente integrados, de manera que, por ejemplo, no es posible aislar el *parser* del motor propiamente; esto hace difícil manejar adecuadamente documentos con acentos o eñes, o incluir modelos de lematización diferentes
- el código, aunque se ha hecho un notable esfuerzo de claridad en su escritura, es prolijo y complejo, lo que hace difícil su modificación. Presenta, además, algunos problemas de portabilidad

3. Objetivos

El objetivo de este trabajo es, pues, la realización de un motor de recuperación experimental, originalmente con la idea de ser utilizado en los trabajos de nuestro propio grupo de investigación. Posteriormente, se pensó que este motor podría resultar útil para otros grupos también; de alguna manera, ésta es la razón de ser del presente informe. Así, las características de partida de nuestro motor son:

- consta tan sólo de dos componentes: un indizador y un estimador de similitud entre consulta y documentos.
- cada uno de estos dos componentes debe aceptar como entrada los resultados de componentes previos, y producir, en su caso, salidas que puedan ser usadas por componentes o procesos posteriores
- el motor debe permitir la utilización, a elección por el usuario, de diferentes algoritmos, e incluir la mayor cantidad de éstos posibles
- el motor debería permitir la inclusión fácil de nuevos algoritmos
- el motor debe permitir la inspección, e incluso la manipulación, de resultados intermedios
- el motor debe estar escrito en un código lo más breve y simple posible, fácil de modificar, e incluso fácil de utilizar como modelo para la realización de otros programas o implementaciones

- dado su carácter experimental, las características de flexibilidad, legibilidad y sencillez de código deberían primar sobre las de eficacia (especialmente velocidad)

Teniendo en cuenta todo esto, se ha considerado una opción razonable la utilización de tablas relacionales para almacenar la información extraída de los documentos, y de sentencias SQL para la realización de las operaciones necesarias [11, 10]. Ambas cosas están lo suficientemente estandarizadas como para poder ser manipuladas o modificadas fácilmente por cualquiera. La estructura relacional y el SQL en particular son especialmente potentes e intuitivas y facilitan la comprensión y ejecución de las operaciones necesarias.

4. El modelo vectorial

El modelo teórico más difundido en RI es el llamado modelo vectorial [19, 17]. Básicamente, según éste, cada documento es representado por un vector D $(d_1, d_2, d_3, ..., d_n)$ donde n es el número de términos posibles en toda la colección de documentos, y cada elemento del vector, en consecuencia, corresponde a cada uno de tales términos. Los elementos del vector, por otra parte, consisten en un valor numérico que trata de expresar la importancia o peso del término en cuestión dentro del documento. Es obvio que un mismo término en documentos diferentes debe tener pesos diferentes.

Las preguntas o consultas se tratan igual que los documentos, y se representan igualmente mediante un vector de pesos. Así, la resolución de una consulta consiste simplemente en la computación de alguna función de similitud entre el vector consulta y cada uno de los vectores de los documentos.

Este tratamiento tiene dos ventajas importantes: una, permite que las consultas se hagan en lenguaje natural, y pueden ser del tamaño que se desee; de hecho, este mecanismo permite usar como consulta otro documento, o incluso comparar documentos entre sí (para categorización o *clustering*, por ejemplo [6]). Y dos, dado que el resultado de la función de similitud no tiene por qué ser binario, es posible establecer una graduación o escala en las respuestas a las consultas, es decir, establecer que unos documentos se adecúan en mayor grado que otros a una consulta determinada [2, 12].

La clave de todo el sistema reside en lo bien que documentos (y consultas) estén representados a través de los vectores; y esto depende de dos factores: la determinación de los términos que se extraen de cada documento, y la forma en que se estiman o calculan los pesos de cada término en cada documento. El primero de estos factores (análisis léxico) queda fuera de nuestro objetivo, pero debe indicarse la conveniencia de aislar esta parte, de forma que, a afectos de experimentación, pueda operarse sobre ella libremente. El segundo factor (el cálculo de los pesos) constituye uno de los elementos centrales de nuestro trabajo.

4.1. El peso de los términos

La estimación del peso de cada término en cada documento puede hacerse de diversas formas, y de hecho se han propuesto una buena cantidad de ellas [16]. Dado que este cálculo ha de hacerse de forma automática, y de manera generalizable para cualquier tipo de documento, los distintos métodos utilizados se basan, de una u otra forma, en las frecuencias de los términos.

El cálculo de los pesos se efectúa a partir de dos factores: la frecuencia de cada término en cada documento, y un elemento conocido como IDF (*Inverse Document Frequency*). Adicionalmente, suele aplicarse algún factor de normalización que permita soslayar las diferencias en tamaño de los documentos (y, en consecuencia, la posibilidad de que las frecuencias sean mayores en documentos más grandes).

El IDF, en líneas generales, es una función inversamente proporcional a la frecuencia del

término en toda la colección de documentos o base de datos; más precisamente, al número de documentos en los cuales aparezca el término. La idea base es que términos que aparezcan en muchos documentos tienen un poder discriminatorio pobre, y viceversa: no tiene mucho sentido, por ejemplo, buscar documentos que contengan el término *ordenador* en una colección de documentos especializados en Informática.

El peso, en consecuencia, podría estimarse a partir de una ecuación genérica:

$$peso\,termino\,en\,documento = \frac{frecuencia\,termino\,en\,documento \times IDF}{factor\,de\,normalizacion}$$

Cada uno de los tres elementos que intervienen en la ecuación puede, a su vez, ser calculado de distintas formas, lo cual da lugar a un gran número de variantes, que suelen conocerse como *esquemas* de pesado. Usualmente, un esquema se representa mediante tres letras, cada una de las cuales identifica la forma en que se han calculado, respectivamente, la frecuencia del término en el documento, el IDF del término y el factor de normalización.

4.2. Esquemas de peso

Entre las muchas posibilidades, las formas más utilizadas de calcular estos tres elementos son:

- La frecuencia del término en el documento:
  - ninguna (**n**):= $n_{td}d$
  - binaria(**b**):= 1
  - max-norm(**m**):= $\frac{n_{td}}{max_{nd}}$
  - aug-norm(**a**):= $0{,}5 + 0{,}5\frac{n_{td}}{max_{nd}}$
  - square(**s**):= n_{td}^2
  - log(**l**):= $ln(n_{td}) + 1$
  - double-log(**d**):= $ln(ln(n_{td}) + 1) + 1$
  - length-norm(**t**):= $\frac{ln(n_{td}+1}{ln(avg_{nd}))+1}$

  donde
  n_{td} es el número de veces que el término t aparece en el documento d
  max_{nd} es el número de veces que aparece el término más frecuente en el documento d
  avg_{nd} es la media de todas las frecuencias de términos en el documento d

- El IDF:
  - none(**n**):= 1
  - tfidf(**t**:= $ln(\frac{N}{n_t})$
  - prob(**p**):= $ln(\frac{N-n_t}{n_t})$
  - frec(**f**):= $\frac{1}{n_t}$
  - square(**s**):= $(ln\frac{N}{n_t})^2$

  donde
  N es el número de documentos en la colección
  n_t es el número de documentos en que aparece el término t

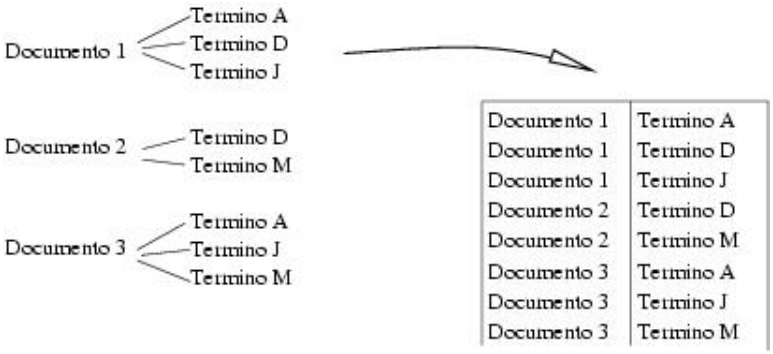

Figura 2: Mapeo de fichero invertido a tabla simple

- Factor de normalización:
  - none(**n**):= 1
  - cosine(**c**):= $\sqrt{\sum_{i=1}^{n} p_{id}{}^2}$
  - sum(**s**):= $\sum_{i=1}^{n} p_{id}$
  - fourth(**f**):= $\sum_{i=1}^{n} p_{id}^4$
  - max(**m**:= max_{pd}

  donde
  n es el número de términos únicos en el documento d
  p_{id} es el peso ($frecuencia \times IDF$) del término i en el documento d
  max_{pd} es el valor más alto de peso sin normalizar en el documento d

5. Implementación

Nuestro motor experimental parte de la idea de que es posible almacenar un fichero invertido procedente de una colección de documentos en una tabla, de manera que, a partir de ahí es posible calcular pesos de términos así como similitudes entre documentos y consultas.

Un fichero invertido, en su forma más básica, no es más que una serie de entradas, una para cada término de la colección de documentos; para cada uno de estos términos se almacena una lista de los documentos en que aparece. Naturalmente, en esa lista pueden almacenarse más cosas (*offset* del documento en que aparece el término, etc,), pero su forma más elemental es la mencionada [9].

Esta estructura puede mapearse simplemente a una tabla (ver figura 2) con dos campos: *término* y *clave de documento*, pero nada impide añadir más columnas para información vinculada a cada una de las parejas término-documento, como frecuencia, *offset*, etc. A partir de aquí es sencillo obtener información adicional para calcular pesos: el número de documentos en la colección (`select count(documento) from tabla;`), el número de documentos en que aparece cada término (`select termino, count(documento) from tabla group by termino;`), etc.

5.1. Estructura básica de la base de datos

La base de datos consta de varias tablas. Algunas podrían ser temporales y desaparecer una vez calculados los pesos, y dependen del esquema de cálculo concreto adoptado en cada ocasión; dado que no son necesarias para la resolución de consultas. Las tablas temporales podrían ser

sustituídas por vistas, solución conceptualmente más elegante, pero que puede dar problemas de rendimiento e impide, en todo caso, la observación y manipulación de resultados intermedios.

Las tablas básicas podrían ser algo como lo siguiente:

- `terminos(termino char(50), documento char(50), veces double)`
- `pesos_def(termino char(50), documento char(50), peso double)`

El tamaño de los campos de tipo `char` podría ser más pequeño, obviamente: términos de 50 caracteres de longitud pueden considerarse directamente errores tipográficos, y las claves de los documentos pueden diseñarse para ocupar bastante menos espacio; incluso los términos podrían ser sustituídos por punteros o claves numéricas, pero eso restaría facilidades de observación.

De otro lado, el número de veces que un término aparece en un documento, que debería ser un entero, se establece como doble. La razón es que esto permite que el *parser* u otro proceso previo aplique, si se desea, algún tipo de coeficiente que prime de distinta forma los términos en función de distintos criterios (lugar del documento donde aparece, tipografía, función sintáctica, etc.)

Para la resolución de consultas, en realidad, sólo es precisa la tabla `pesos_def`, pero la tabla `terminos` puede ser interesante conservarla para posibles recalculados de pesos posteriores.

5.2. Entrada de datos

El motor de recuperación al estar aislado del *parsing* de documentos [8], recibe como entrada el resultado de éste, en la forma de

```
"término","documento",número de ocurrencias en documento
```

y, tal cual se almacena en una tabla. Sobre la información almacenada en esta tabla se harán las operaciones posteriores.

Aquí no se efectúa ningún chequeo ni ninguna otra operación previa sobre los datos de entrada. Esto significa que cosas como la normalización de caracteres, eliminación de palabras vacías, etc. es responsabilidad del *parser* o de otros cualesquiera procesos intermedios que se quieran añadir.

5.3. Cálculo de pesos

El cálculo de pesos puede efectuarse en tres fases, una para cada componente del peso. Cada una de estas fases termina con una tabla temporal que recoge el componente calculado y que es usada en la fase siguiente; la tercera y última fase finaliza con la consecución de la tabla `pesos_def`, con lo que esas tablas temporales dejan de ser necesarias.

Obsérvese que el sistema aquí empleado en tres fases puede resultar, en ocasiones, innecesariamente costoso. De alguna manera, ésta es una de las cosas que distingue un motor experimental de uno operacional; en uno de éstos, que aplica un esquema o algoritmo determinado, se puede ir directamente a él. Por ejemplo, para operar con un esquema de pesos `ntc` (el más frecuente), no es preciso calcular frecuencia del término en el documento, ni almacenarla en ninguna tabla intermedia. Pero, como, en un motor experimental, podemos encontrarnos con cualquier esquema, y las combinaciones posibles son numerosas, parece más sensato y más manejable operar en esas tres fases.

5.3.1. La frecuencia del término en el documento

Hay diversas maneras de estimar este componente del peso, y que no tiene por qué coincidir con el número de veces que cada término aparece en cada documento; en cualquier caso, el

objetivo, en esta fase, es doble: por un lado, terminar obteniendo una tabla temporal con estas frecuencias, calculadas en la forma en que se desee, que pueda ser utilizada en las sucesivas fases del cálculo de pesos. Por otro, mantener los datos de entrada originales en su propia tabla, de forma que puedan ser reutilizados (por ejemplo, para recalcular pesos con otro esquema distinto).

Así, la base de esta fase es una simple sentencia SQL:
`Create table frecuencias as select ...`
El contenido concreto depende del esquema de cálculo aplicado, pero, en general, se tratará de un `select` sobre la tabla `terminos` que contiene los datos originales de entrada. Con algunos esquemas, que utilizan, por ejemplo, cosas como la frecuencia máxima en el documento, es preciso algún paso intermedio que calcule tales elementos.

Para los esquemas más conocidos, la solución requiere básicamente las siguientes sentencias SQL:

- **n**one:

```
create table frecuencia as
select termino as termino,
documento as documento, frecuencia as frecuencia
from terminos;
```

- **b**inary:

```
create  table frecuencia as
select termino as termino, documento as documento,
1 as frecuencia
from terminos;
```

- **m**ax-norm:
  Este esquema requiere la obtención previa del número de veces que aparece el término más repetido en cada documento. Obsérverse que este dato es el mismo para todos los términos de un mismo documento:

```
create  table max_frec as
select documento as documento, max(frecuencia) as frecuencia
from terminos group by documento;
```

  Una vez obtenidos los máximos para cada documento,

```
create  table frecuencia as
select terminos.termino as termino,
terminos.documento as documento,
divide(terminos.frecuencia,max_frec.frecuencia)
as frecuencia
from terminos, max_frec
where terminos.documento=max_frec.documento;
```

- **a**ug-norm:
  como en el caso anterior, necesitaremos los máximos y, una vez calculados:

```
create  table frecuencia as
select terminos.termino as termino, terminos.documento
as documento,
0.5+0.5*(divide(terminos.frecuencia,max_frec.frecuencia))
as frecuencia
from terminos, max_frec
where terminos.documento=max_frec.documento;
```

- **s**quare:

```
create  table frecuencia as
select termino as termino, documento as documento,
frecuencia*frecuencia as frecuencia
from terminos;
```

- **l**og:

```
create  table frecuencia as
select termino as termino, documento as documento,
ln(frecuencia)+1 as frecuencia
from terminos;
```

Obviamente, se deberían construir índices sobre los campos `termino` y `documento` para agilizar las operaciones.

5.3.2. El IDF y el peso sin normalizar

El IDF es el mismo para cada término, independientemente de en qué documento aparezca éste. De manera que el resultado básico del cálculo del IDF podría ser una tabla con los campos `termino` e `idf`. El peso sin normalizar, por otra parte, es el resultado de multiplicar frecuencia por IDF; así, obtenido el IDF, puede obtenerse en la misma fase el peso sin normalizar. El producto final de esta fase es, en consecuencia, una tabla con los campos `termino`, `documento` y `peso`.

Con los esquemas más frecuentes, el IDF se calcula con las siguientes sentencias SQL:

- **n**one:

```
create  table idf as
select termino as termino, 1 as idf
from terminos group by termino;
```

- **t**fidf:
  Este esquema requiere conocer el número de documentos que hay en toda la colección, al cual llamaremos N, y que puede ser obtenido mediante sentencias simples, tal como se ha mencionado más arriba. Este tipo de datos, simples y únicos para toda la colección, se calculan en nuestro proyecto al principio de las operaciones y se almacenan en variables. Una vez conocido N,

```
create table idf as
select termino as termino,
ln(N/count(documento)) as idf,
from terminos
group by termino;
```

- **p**rob Este esquema, al igual que el anterior, también requiere N:

```
create  table idf as
select termino as termino,
ln(N-count(documento)/count(documento))as idf,
from terminos
group by termino;
```

- **f**req:

```
create  table idf as
select termino as termino,
1/count(documento) as idf,
from terminos group by termino;
```

- **s**quared:

```
create  table idf as
select termino as termino,
ln(N/count(documento)^2) as idf,
from terminos group by termino;
```

Una vez obtenido el IDF, sólo nos queda calcular el peso sin normalizado mediante una simple multiplicación, y almacenarlo en una tabla:

```
create table pesos as
select frecuencia.termino as termino,
frecuencia.documento as documento,
frecuencia.frecuencia*idf.idf as peso
from frecuencia,idf
where frecuencia.termino=idf.termino;
```

Obviamente, habrá sido preciso crear índices en la tabla `idf` para mayor rapidez, y también en `pesos` para la fase siguiente. Obsérvese que la tabla `frecuencia` ya no es necesaria y podría eliminarse. No ocurriría lo mismo con la tabla `idf`, que sería necesaria para estimar los pesos de los términos de las consultas, si éstos se calculasen con el mismo esquema que los de los documentos.

Pero no siempre es así, y una de las razones de ser de un motor experimental es poder aplicar esquemas distintos en documentos y consultas; de manera que parece que un código más estructurado y legible sugiere tratar las consultas de forma independiente, y no presuponer el uso de ningún esquema determinado de antemano. Desde este punto de vista, la tabla `idf` tampoco sería ya necesaro conservarla.

5.3.3. El factor de normalización y pesos definitivos

Esta fase, última por lo que se refiere a los documentos, requiere el cálculo de un factor de normalización, y la posterior división del peso sin normalizar que acabamos de almacenar en la tabla `pesos` por dicho factor. El factor de normalización, por otra parte, es único para cada documento.

Para los esquemas más habituales, obtendríamos dicho factor mediante las siguientes sentencias, según cada caso:

- **n**one:

```
create table sumatorios as
select documento as documento, 1 as sumatorio
from terminos
group by documento;
```

- **c**osine:

```
create table sumatorios as
select documento as documento,
SQRT(sum(peso*peso)) as s
from pesos group by documento;
```

- **s**um:

```
create table sumatorios as
select documento as documento,
sum(peso*peso) as s
from pesos group by documento;
```

- **f**ourht:

```
create table sumatorios as
select documento as documento,
sum(peso^4) as s
from pesos group by documento;
```

- **m**ax:

```
create table sumatorios as
select documento as documento,
max(peso) as s
from pesos group by documento;
```

- pivoted **u**nique norm:
  Este esquema utiliza dos elementos, *pivot* y *slope*, además del número de términos en cada documento. *pivot* es el número medio de términos únicos por documento, y se trata, obviamente, de un valor único para toda la colección, que se obtiene mediante una simple sentencia SQL, tal como se comentó anteriormente.

  El *slope* es un coefiente cuyo valor debe fijarse a voluntad; experimentalmente se ha demostrado que los mejores resultados se obtienen con valores entre 0.1 y 0.3.

  El número de términos en cada documento se obtiene y se almacena en una tabla `num_terms`:

```
create table num_terms as
select documento as documento,
count(termino) as n_terms
from terminos group by documento;
```

  Con estos datos, el factor de normalización se obtiene mediante:

```
create table sumatorios as
pesos.documento as documento,
pesos.peso,(1-slope)*pivot+slope*num_terms.n_terms as s,
from pesos, num_terms
where pesos.documento=num_terms.documento;
```

  Naturalmente, la tabla `num_terms` ya no se necesita más, después de esto.

Una vez obtenido el factor de normalización, sólo queda normalizar los pesos y almacenarlos en una tabla `pesos_def`:

```
create table pesos_def as
select pesos.termino as termino,
pesos.documento as documento,
pesos.peso/sumatorios.s as peso,
from pesos, sumatorios
where pesos.documento=sumatorios.documento;
```

Hasta aquí, hemos obtenido los pesos de los términos de los documentos, con lo que sólo necesitamos la tabla `pesos_def` y la que contiene los datos originales, `terminos`, de manera que podemos deshacernos de las demás. La tabla `pesos_def`, por otra parte, requeriría un índice de `termino`, para resolver más rápidamente las consultas.

5.4. Pesos de las consultas

Los pesos de los términos de las consultas pueden estimarse de la misma manera que los de los documentos, aunque, como ya se ha dicho, aplicando esquemas que no tienen por qué ser iguales.

El hecho de que, al procesarse una sola consulta de cada vez, el volumen de información involucrado sea mucho menor, puede aconsejar buscar métodos más ágiles para calcular los pesos de los términos de las consultas. De hecho, en sistemas interactivos las consultas suelen ser muy cortas (2 ó 3 términos); en muchos sistemas ni siquiera se calculan pesos para las consultas.

Una posibilidad, dependiente del rendimiento del sistema SQL que se utilice es el uso de vistas; esto nos permite, al tiempo que se indiza la colección de documentos, dejar construidas las vistas necesarias para calcular los pesos de los términos de las consultas. En el momento de la consulta, estas vistas se ejecutan, obteniendo los pesos correspondientes.

De una forma u otra, el producto de este proceso es una tabla o vista (`pesos_c`) con dos campos: `termino` y `peso`.

5.5. Resolución de consultas

La resolución de consultas es simple, una vez que tenemos los pesos de los términos. Básicamente, se trata de localizar los documentos con algún término en común con la consulta y, una vez localizados, calcular un coeficiente de similitud entre cada uno de esos documentos y la consulta. Posteriormente, ordenaremos esos documentos de forma decreciente en función de ese coeficiente de similitud.

El coeficiente de similitud más habitual consiste en el producto de los vectores [17]. Dado que los pesos en esos vectores han sido normalizados previamente, podemos resolver una consulta mediante una sentencia como la que sigue:

```
select pesos_def.documento,
sum(pesos_def.peso*pesos_c.peso) as simil
from pesos_def, pesos_c
where pesos_def.termino=pesos_c.termino
group by pesos_def.documento order by simil DESC ;
```

6. Conclusiones

Los motores de recuperación para la investigación experimental son importantes instrumentos para la experimentación, que permiten aplicar diferentes algoritmos y observar sus resultados. En su diseño es más importante la modularidad y la facilidad de modificar procedimientos que la rapidez en la resolución de las consultas; la posibilidad de observar y modificar resultados de pasos intermedios es también importante [7].

La utilización de un sistema de base de datos relacional y sentencias SQL permite construir un sistema de Recuperación de Información para la investigación experimental. Se ha mostrado cómo mediante sencillas sentencias SQL pueden implementarse diferentes algoritmos de pesado de términos; los resultados de cada uno de los pasos intermedios pueden almacenarse en tablas para ser observados e incluso modificados.

Al mismo tiempo, los tiempos de ejecución, sin ser especialmente brillantes, resultan lo suficientemente ágiles como para utilizarse incluso en entornos reales.

Referencias

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, Harlow, England, 1999.

[2] N. J. Belkin and W. B. Croft. Retrieval techniques. *Annual Review of Information Science and Technology*, 22:109–145, 1987.

[3] C. Buckley. New retrieval approaches using smart: Trec-4. In D. Harman, editor, *The Fourth Text REtrieval Conference (TREC-4)*, number 4, pages 25–48, Gaithersburg, Maryland,

noviembre 1995. National Institute of Standards and Technology (NIST), Defense Advanced Research Projects Agency (DARPA).

[4] C. Buckley, J. Allan, and G. Salton. Automatic routing and ad-hoc retrieval using SMART: TREC 2. In *The Second Text REtrieval Conference (TREC-2)*, pages 35–44. NIST Special Publication 500-215, 1993.

[5] C. Buckley, G. Salton, J. Allan, and A. Singhal. Automatic query expansion using smart: Trec 3. In D. Harman, editor, *Overview of the Third Text REtrieval Conference (TREC-3)*, number 3, pages 69–80, Gaithersburg, Maryland, noviembre 1994. National Institute of Standards and Technology (NIST), Advanced Research Projects Agency (ARPA).

[6] S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan Kaufmann Publishers, 2002.

[7] C. G. Figuerola, J. L. Alonso Berrocal, and Á. F. Zazo Rodríguez. Diseño de un motor de recuperación de información para uso experimental y educativo. *BID - Textos universitaris de bibliotecomia i documentació*, 4, 2000. Electronic Publication: http://www.ub.es/biblio/bid/bid04.htm.

[8] C. Fox. Lexical analysis and stoplist. pages 102–130. Prentice-Hall Inc., Englewood Cliffs (NJ), 1992.

[9] W. B. Frakes. Introduction to information storage and eetrieval systems. pages 1–12. Prentice-Hall Inc., Englewood Cliffs (NJ), 1992.

[10] D. Grossman, O. Frieder, D. Holmes, and D. Roberts. Integrating structured data and text: A relational approach. *JASIS*, 48(2), febrero 1997.

[11] D. A. Grossman, C. Lundquist, J. Reichart, D. Holmes, A. Chowdhury, and O. Frieder. Using relevance feedback within the relational model for trec-5. In D. Vorhees, E.M.; Harman, editor, *The Fifth Text REtrieval Conference (TREC-4)*, number 5, pages 405–414, Gaithersburg, Maryland, noviembre 1996. National Institute of Standards and Technology (NIST), Defense Advanced Research Projects Agency (DARPA).

[12] D. K. Harman. Ranking algorithms. pages 363–392. Prentice-Hall Inc., Englewood Cliffs (NJ), 1992.

[13] V. Raghavan, P. Bollmann, and G. S. Jung. A critical investigation of recall and precision as measures of retrieval system performance. *ACM Transactions on Information Systems*, 7(3):205 – 229, julio 1989.

[14] C. v. Rijsbergen. *Information retrieval*. 1979.

[15] G. Salton. *The SMART Retrieval System. Experiments in Automatic Document Processing*. Prentice Hall, Englewoods Cliffs, N. J., 1971.

[16] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[17] G. Salton and M. McGill. *Automatic Text Processing*. Adisson-Wesley, Reading, 1989.

[18] SearchTools.com. Search tools product listings in alphabetical order, 2005.

[19] A. F. Zazo, C. G. Figuerola, J. L. A. Berrocal, and R. Gómez. Recuperación de información utilizando el modelo vectorial. participación en el taller CLEF-2001. Technical Report DPTOIA-IT-2002-006, Departamento de Informática y Automática - Universidad de Salamanca, Mayo 2002. On line: http://tejo.usal.es/inftec/2002/DPTOIA-IT-2002-006.pdf.