

UNIVERSIDAD DE CASTILLA-LA MANCHA  
ESCUELA SUPERIOR DE INFORMÁTICA



# Almacenamiento y Recuperación de la Información

PYDMS - YET ANOTHER DOCUMENT MANAGEMENT  
SYSTEM

13 de diciembre de 2009

Juan Andrada Romero  
Jose Domingo López López

# ÍNDICE

<b>1. Introducción</b>	<b>1</b>
<b>2. Decisiones de diseño</b>	<b>1</b>
2.1. Lenguaje de programación y sistema operativo elegido . . . . .	1
2.2. Diseño del sistema . . . . .	1
2.2.1. Diseño multicapa . . . . .	1
2.2.2. Diagrama de clases . . . . .	2
2.3. Diseño de la base de datos . . . . .	3
2.4. Desarrollo del sistema . . . . .	5
2.4.1. Motor de indexación . . . . .	6
2.4.2. Motor de búsqueda . . . . .	7
<b>3. Manual de usuario</b>	<b>9</b>
3.1. Instalación . . . . .	9
3.2. Motor de indexación . . . . .	9
3.2.1. Interfaz gráfica de usuario . . . . .	9
3.2.2. Interfaz de usuario por línea de comandos . . . . .	10
3.3. Motor de búsqueda . . . . .	10
<b>4. Estadísticas de funcionamiento</b>	<b>15</b>
4.1. Motor de indexación . . . . .	15
4.2. Motor de búsqueda . . . . .	15
<b>Referencias</b>	<b>17</b>

## 1. INTRODUCCIÓN

Este documento aborda el sistema de almacenamiento y recuperación de la información que ha sido desarrollado. En la sección 2 se tratarán las decisiones de diseño tomadas a la hora de desarrollar el sistema, así como el diseño de la base de datos y un diagrama de clases que detalla este sistema. En la sección 3 se explica brevemente el funcionamiento de esta aplicación, mientras que en la sección 4 se comentan algunas estadísticas de funcionamiento del sistema.

## 2. DECISIONES DE DISEÑO

En las siguientes subsecciones se van a detallar las decisiones de diseño que se han tenido en cuenta a la hora de desarrollar e implementar este sistema. Algunas de estas decisiones de diseño son las mismas que se tomaron cuando se desarrolló el módulo de indexación de documentos, aunque se han realizado algunos cambios.

### 2.1. Lenguaje de programación y sistema operativo elegido

Para implementar el sistema, se ha decidido utilizar el lenguaje de programación Python ([5]). Se ha seleccionado este lenguaje de programación ya que permite el uso de estructuras como diccionarios (tablas hash), listas, manejadores de archivos, llamadas al sistema, definición de patrones mediante expresiones regulares, etc. Dichos elementos han facilitado el desarrollo del sistema, ya que Python los gestiona de una manera eficaz y permite su uso de una manera sencilla.

Por otra parte, se ha elegido un sistema UNIX porque resulta más cómodo a la hora de llevar a cabo la implementación.

### 2.2. Diseño del sistema

#### 2.2.1. *Diseño multicapa*

El sistema se ha desarrollado siguiendo el enfoque multicapa, para desacoplar las operaciones del dominio de las operaciones de persistencia y presentación, consiguiendo así un sistema extensible y reutilizable.

Una muestra de reutilización es que las clases de la capa de presentación conocen a las clases de dominio, pero no al revés, por lo que podría cambiarse la interfaz de usuario sin tener que cambiar las clases de dominio.

Por otra parte, el dominio es totalmente independiente de la persistencia, pues se implementa el patrón DAO (también llamado Fabricación Pura), encargado de gestionar la persistencia, tanto en la base de datos como en un fichero. Además, el DAO que se usa para almacenar la información en la base de datos, utiliza a su vez un patrón Agente Singleton, encargado de inicializar la conexión de la base de datos si no existe ya una instancia del agente, cerrar la conexión y ejecutar las sentencias SQL que recibe del patrón DAO.

De este modo, el patrón DAO se ha utilizado para desacoplar el dominio de la persistencia, evitando así que las propias clases del dominio creen las sentencias SQL necesarias para gestionar la base de datos, pasando ésto a ser responsabilidad del patrón DAO. Por tanto, si en algún momento se cambia el sistema gestor de base de datos, las clases de dominio no se verían afectadas, ya que sólo habría que cambiar el Agente y el DAO.

Esta división en capas y las diferentes clases diseñadas e implementadas se detallan en el apartado siguiente.

### 2.2.2. Diagrama de clases

En la Figura 2.1 se muestran las clases que conforman el sistema. Como puede observarse en la Figura 2.1, y como se mencionó en el apartado anterior, se han diseñado tres capas (representadas por paquetes) que permiten desacoplar unas clases de otras, ya que sólo se utilizan dependencias de uso entre clases.

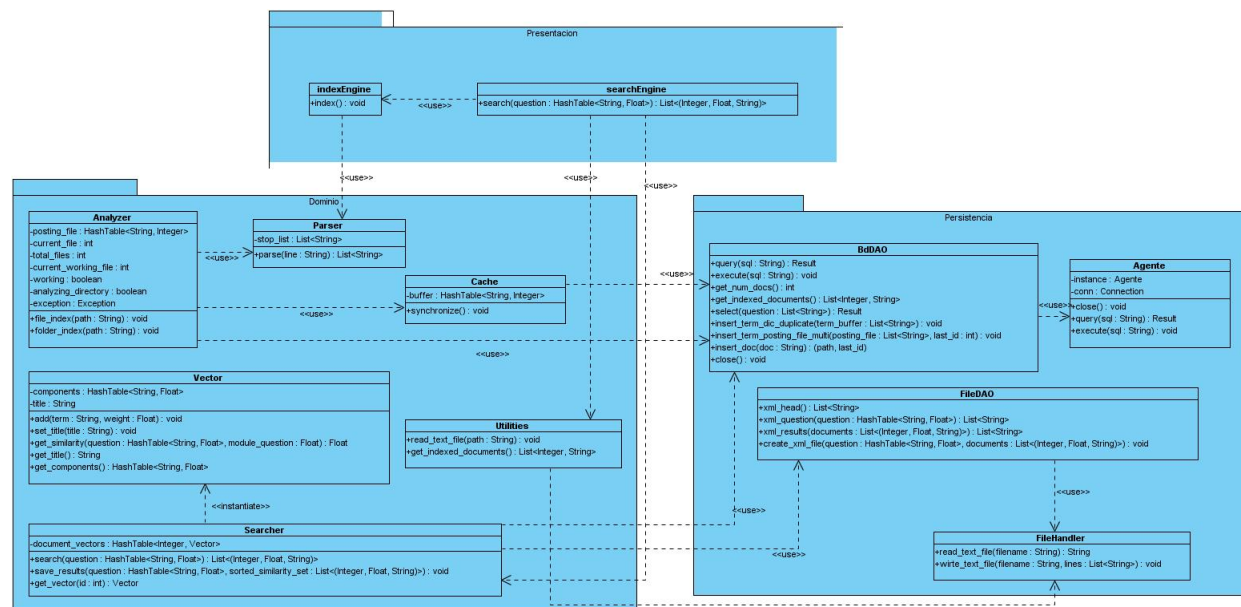


Figura 2.1: Diagrama de clases del sistema desarrollado

A continuación, se explica brevemente la responsabilidad de algunas de las clases más importantes.

**Capa de dominio** En esta capa se encuentran las clases que son necesarias para soportar las funcionalidades que el sistema requiere.

- **Analyzer:** se encarga de leer el fichero o los ficheros del directorio indicado en la ruta introducida en la interfaz gráfica. A continuación, para cada fichero, utiliza al módulo *parser* (que se detallará posteriormente) para obtener los términos de ese documento, hace una copia de dicho documento en el repositorio del sistema e inserta su título y ruta en la base de datos, utilizando al módulo *bdDao*.
- **Cache:** se utiliza para ir almacenando en memoria términos, intentando disminuir el número de accesos a disco. Se explica más en detalle en un apartado posterior.

- **Vector**: se utiliza para crear los vectores de cada documento cuando se realiza una búsqueda, indicando el título del documento, los términos del documento y sus pesos. La función `get_similarity` calcula la semejanza entre la pregunta introducida y el vector documento (ver ecuación (2.1)). El módulo *searcher* (se detalla posteriormente) es el que va creando estos vectores al buscar documentos que tengan algún término en común con la pregunta dada.
- **Utilities**: es una clase intermedia que se encuentra en la capa de dominio, que ofrece a la capa de presentación la posibilidad de leer un fichero de texto y consultar los documentos que están indexados en el sistema.

Además de estas clases, se cuenta también con el módulo **exception**, que contiene excepciones personalizadas que lanzan las clases anteriores, y el módulo **config**, que contiene la definición de constantes que, de algún modo, configuran ciertos aspectos del sistema (rutas de directorios y tamaño de la caché).

**Persistencia** En cuanto a la capa de persistencia, las clases que se encuentran son:

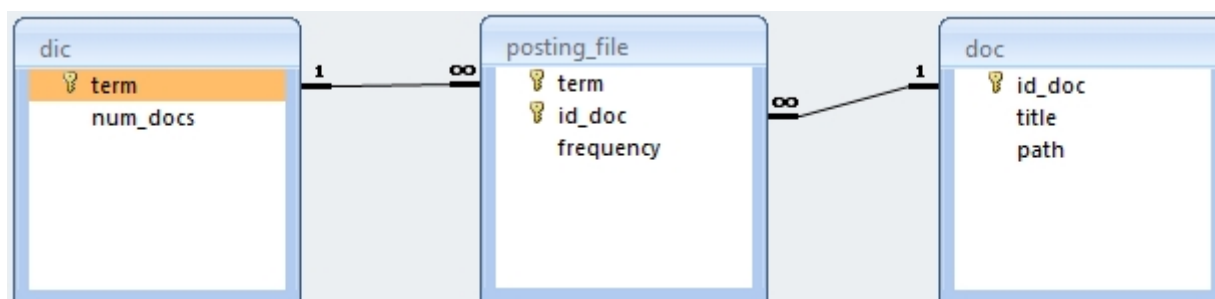
- **BdDao**: construye las sentencias SQL con los datos de las clases del dominio y delega su ejecución en el Agente. El objetivo de esta clase es separar las capas de dominio y persistencia.
- **FileDao**: se encarga de construir la estructura del fichero XML que se genera después de cada búsqueda. Delega la escritura de éste a la clase *fileHandler*, que es la encargada de leer y escribir ficheros en disco.

**Presentación** En esta capa se encuentran las interfaces gráficas de usuario para el motor de indexación y de búsqueda, cuyo funcionamiento se detalla en la sección 3.

## 2.3. Diseño de la base de datos

Como sistema gestor de base de datos se ha optado por utilizar MySQL, ya que es ligero, libre, gratuito (ya que no se destina a fines comerciales) y compatible con otros sistemas, como Windows.

Por otra parte, como ya se comentó en la documentación del motor de indexación, el diseño de la base de datos era el que se muestra en el diagrama de la Figura 2.2.



**Figura 2.2:** Diagrama de la base de datos antigua

Sin embargo, ha sido necesario modificar ligeramente este diseño, para optimizar las búsquedas y consumir menos tiempo. El cambio que se ha introducido es simplemente un cambio de la clave primaria de la tabla *posting\_file*, pasando a tener una clave autonumérica. Con esto (y optimizando

las sentencias SQL, como se detallará posteriormente), se ha reducido tanto el tiempo de búsqueda, como el de indexación.

Así, el nuevo diagrama de tablas se muestra en la Figura 2.3, mientras que las tablas que existen en el sistema son:

- **dic**: implementa el diccionario de términos. Tiene los campos *term* y *num\_docs*, que representan cada uno de los términos encontrados, junto con el número de documentos donde aparece cada término. La clave primaria es el término.
- **doc**: implementa la tabla de documentos. Tiene los campos *id\_doc*, *title* y *path* para almacenar un identificador único de documento, el título del documento y la ruta del sistema donde se almacena una copia de dicho documento.
- **posting\_file**: implementa el posting\_file. Contiene los campos *id*, *term*, *id\_doc* y *frequency* para representar la frecuencia con la que aparece un término en un documento. Por tanto, el campo *term* hace referencia a términos del diccionario, y el campo *id\_doc* hace referencia a los documentos de la tabla de documentos.

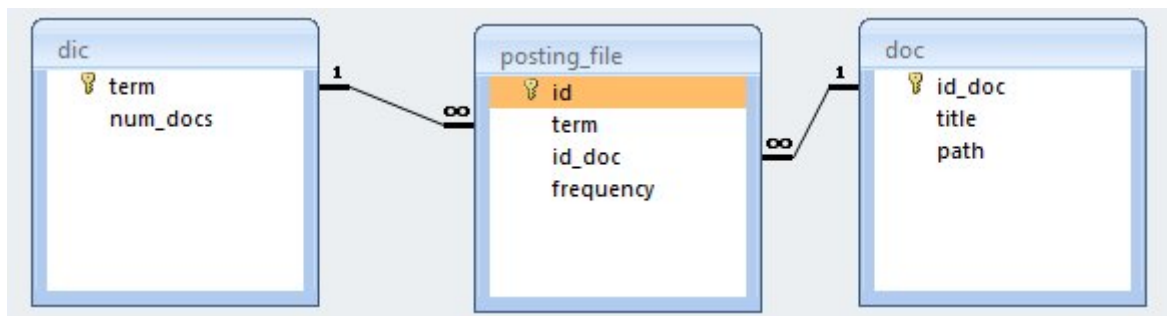


Figura 2.3: Diagrama de la nueva base de datos

**Optimización de las sentencias SQL en el indexador de documentos** En la primera versión que se desarrolló del módulo encargado de indexar los documentos, los términos que se iban encontrando en un documento, se iban insertando uno a uno en la tabla *posting\_file* y en el diccionario, además de ir actualizando la frecuencia de aparición del diccionario, con una sentencia **update**. Esto hacía que el acceso a la base de datos ocupase gran parte del tiempo, ya que se realizaban numerosas sentencias **insert** y **update**, para actualizar la tabla *posting\_file* y la tabla *dic* con los términos que se iban parseando.

Estudiando la sintaxis completa de la sentencia **insert** del manual de MySQL, se decidió hacer una inserción de múltiples filas en la tabla *posting\_file*, donde cada fila corresponde a cada uno de los términos encontrados en un documento (junto con el identificador del documento y su frecuencia). Así, la sentencia **insert** que se utiliza es similar a la siguiente, que aparece en el manual de MySQL:

```
INSERT INTO tbl_name [(col_name,...)] VALUES (expr),(expr), ...
```

Del mismo modo, la sentencia **insert** permite hacer cambios en una columna cuando se intenta insertar una fila con clave primaria duplicada. Por tanto, esto puede utilizarse para realizar inserciones múltiples y, a la vez, actualizaciones en el diccionario, ya que si el término no se encontraba en el diccionario, se insertará, y si ya se encontraba, ocurrirá un fallo de clave duplicada (porque la clave primaria de la tabla *dic* es el término) y se actualizará con la frecuencia correspondiente. La sentencia utilizada para insertar y actualizar términos en el diccionario es el siguiente:

```
INSERT INTO tbl_name [(col_name,...)] VALUES (expr),(expr), ...  
ON DUPLICATE KEY UPDATE num_docs=num_docs+VALUES(num_docs)
```

**Optimización de las sentencias SQL en la búsqueda de documentos** En un primer momento, para obtener todos los datos (frecuencias, identificador del documento, título, etc.) de los documentos que tenían términos en común con la pregunta, se pensó en hacer varias sentencias **select** por separado, pero esto consumía demasiado tiempo para una búsqueda.

Por tanto, se optó por hacer una única consulta componiendo las tres tablas y obteniendo así todos los datos necesarios para calcular los pesos y la relevancia. Esta consulta es la que se muestra a continuación:

```
SELECT posting_file.term,posting_file.id_doc,frequency,num_docs,title  
FROM posting_file JOIN dic JOIN doc  
ON dic.term=posting_file.term AND posting_file.id_doc IN  
(SELECT DISTINCT p1.id_doc FROM posting_file p1 WHERE  
p1.term IN (lista de términos de la pregunta))
```

Esa sentencia aún tardaba demasiado tiempo en procesarse, por lo que era necesario realizar unos cambios. Estudiando el manual de la sentencia **select** de MySQL, se observó que se puede elegir el índice (clave ajena) con el cuál combinar las tablas, lo que puede acelerar la búsqueda. Además, se puede indicar que esa sentencia tenga alta prioridad, ejecutando la sentencia en la base de datos antes que cualquier otra. Por otro lado, se optó también por crear una vista temporal con los resultados de la subconsulta anterior, para luego combinar con esa vista, en vez de tener una consulta con una subconsulta. Así, obtenemos estas sentencias:

```
CREATE VIEW view_name AS SELECT DISTINCT p1.id_doc  
FROM posting_file p1 USE INDEX (term) WHERE  
p1.term IN (lista de términos de la pregunta)  
  
SELECT HIGH_PRIORITY posting_file.term, posting_file.id_doc, frequency,  
num_docs, title  
FROM posting_file USE INDEX (term,id_doc) JOIN view_name v JOIN dic  
JOIN doc  
ON posting_file.id_doc=v.id_doc AND dic.term=posting_file.term  
AND posting_file.id_doc=doc.id_doc
```

De este modo, se han conseguido tiempos mucho más pequeños que en el caso anterior, como se muestra en la sección 4.

## 2.4. Desarrollo del sistema

Para terminar, en esta subsección se detallan el motor indexador de documentos y el de búsqueda, comentando las decisiones tomadas.

Cabe destacar que en la aplicación se utiliza el módulo *psyco*, que es un módulo externo que realiza una compilación *just-in-time* y que incrementa la velocidad de ejecución del código Python (ya que éste no se interpretará). Gracias a este módulo el tiempo de ejecución de la aplicación se puede reducir hasta la mitad.

### 2.4.1. Motor de indexación

**Conexión con la base de datos** En una primera iteración del desarrollo de este módulo, la conexión con la base de datos se creaba cada vez que se quería acceder a ella, como, por ejemplo, al actualizar la frecuencia de un término en el *posting\_file*, cerrándose al terminar el acceso. Esto consumía mucho tiempo y el acceso a disco era muy elevado, por lo que se ha optado por inicializar la conexión con la base de datos al lanzar el módulo de indexación, y cerrarla cuando termina.

**Copia de documentos** Cuando se analiza un documento para realizar su indexación, en un primer momento se copiaba el documento en la base de datos documental del sistema leyendo línea por línea. Este proceso era algo lento, por lo que finalmente se ha utilizado una llamada al sistema implementada en C, consiguiendo que la copia del documento completo sea instantánea.

**Parser** Para realizar el tratamiento de los documentos y prepararlos para su indexación, se ha implementado el módulo *parser*. Dicho módulo recibe una línea y devuelve una lista con todos los términos contenidos en esa línea. Para ello, realiza las siguientes acciones:

- En primer lugar, las palabras de la línea se convierten a minúsculas y se escapan los caracteres “\” y “ ” de las palabras que los contengan, ya que si se intenta insertar un término con esos caracteres en la base de datos, la sentencia sql no se forma de manera correcta y se provoca una excepción al ejecutar esa sentencia en la base de datos.
- Se definen separadores de palabras, que son tanto los signos de puntuación, como los espacios en blanco (incluyendo saltos de línea, tabuladores, etc). Se define también un patrón para detectar direcciones IP y números.
- Todos los símbolos separadores que estén contenidos en la línea (menos el punto y el guión) se reemplazan por un espacio en blanco y se divide la línea en palabras, separando por espacios, obteniendo así una lista de palabras, que recibe el siguiente tratamiento:
  1. Si una palabra es un espacio en blanco o se encuentra en la *stop\_list*, se ignora.
  2. Si una palabra se corresponde con el patrón IP, se almacena esa palabra en la lista de palabras que va a devolver el parser.
  3. Si una palabra se corresponde con el patrón de un número, también se almacena ese número.
  4. Se reemplazan las vocales acentuadas de las palabras por vocales sin acentuar.
  5. Se reemplazan los puntos contenidos en la palabra por espacios en blanco. Se hace ahora y no antes para no perder los puntos que contiene una dirección IP.
  6. Si una palabra tiene un guión, esa palabra se almacena con el guión y por separado, siempre y cuando las palabras por separado no estén en la *stop\_list*. Por ejemplo, si aparece *cd-rom*, se almacenarían las palabras “cd-rom”, “cd” y “rom”.
  7. Una palabra que no comience con un carácter alfanumérico, se ignora.

Con este tratamiento, sólo se almacenan direcciones IP, números y palabras, separando en palabras cualquier ruta, dirección web o e-mail.



**Codificación de documentos y de la base de datos** Tras realizar múltiples pruebas con diferentes documentos, se observaron fallos a la hora de recuperar y almacenar términos en la base de datos. Estos fallos eran debidos a la codificación de los documentos, que debe ser UTF-8 para el correcto funcionamiento en sistemas UNIX. Por tanto, todos los documentos de la colección se han estandarizado a la codificación UTF-8, al igual que las diferentes tablas de la base de datos.

Aunque la codificación de la base de datos sea UTF-8, se han realizado pruebas con documentos escritos en codificación *latin-1*, y la indexación funciona correctamente, aunque es algo más lenta y a veces aparecen términos con caracteres extraños, debido a dicha codificación.

**Posting\_file** Se ha optado por almacenar el `posting_file` de cada documento en memoria en forma de diccionario (tabla hash), para conseguir que los accesos sean más rápidos. Se han realizado pruebas y para un documento de unas 25.000 líneas, el tamaño del `posting_file` no excede de los 13MB en memoria. Por tanto, esta opción es más eficiente en cuanto a tiempo, ya que sólo se vuelca el `posting_file` a la base de datos cuando se termina de procesar cada documento, en lugar de ir accediendo a la base de datos por cada uno de los términos que se deben almacenar.

Otra de las razones para hacer esto es para poder aprovechar la sentencia **insert** múltiple que se comentó en el apartado 2.3.

**Memoria caché** Para disminuir el número de inserciones/actualizaciones en la tabla diccionario, se ha decidido implementar una caché (en forma de tabla hash), donde se van almacenando términos, junto con el número de documentos en los que aparece dicho término. Así, cuando la caché está llena, se vacía a la base de datos, con la sentencia **insert** que se detalló en el apartado 2.3. También se vacía antes de insertar los términos de un documento en la tabla `posting_file`, para mantener la integridad de clave ajena que dicha tabla tiene hacia el diccionario.

### 2.4.2. Motor de búsqueda

**Searcher** Este módulo es el encargado de realizar la búsqueda de los documentos relevantes a una pregunta dada, siguiendo el modelo vectorial. Se ha implementado este modelo ya que no está sujeto a las limitaciones del modelo booleano y da unos resultados similares al modelo probabilístico siendo más sencilla su implementación. En cuanto a la aproximación implementada, se ha elegido la que calcula la semejanza de dos vectores en función del ángulo que forman, ya que es más precisa que la que calcula la distancia entre dos vectores. Esta segunda aproximación es más ineficiente debido a que los usuarios suelen buscar entre 3 y 5 términos de media y los documentos con pocos términos serían muy próximos a la pregunta.

Por tanto, la fórmula que se sigue en este modelo para calcular la relevancia entre un documento y la pregunta es la mostrada en la ecuación (2.1).

$$sem(p, d_i) = \cos(\alpha) = \frac{\sum_{j=1}^n wp_j w_{ij}}{\sqrt{\sum_{j=1}^n wp_j^2} \sqrt{\sum_{j=1}^n w_{ij}^2}} \quad (2.1)$$

El funcionamiento de este módulo es el siguiente:

1. Se aplica el módulo *parser* a la pregunta que el usuario introduce, obteniendo una lista de términos con el peso introducido (1, por defecto).
2. Se calcula el módulo de la pregunta, a partir de los pesos de cada uno de sus términos.
3. Se consultan los términos y frecuencias de los documentos que tienen algún término en común con la pregunta, utilizando la sentencia **select** que se comentó en el apartado 2.3.
4. Se recorren las filas que la consulta devuelve, creando los vectores de los documentos encontrados. Dicho vector contiene el término y su peso, que se calcula con la ecuación (2.2).
5. Se recorren los vectores de los documentos, para calcular su semejanza con la pregunta.
6. Finalmente, se devuelve una lista con la información de los documentos ordenados de mayor a menor semejanza, además de escribir la información en el fichero XML.

De este modo, al mismo tiempo que se van recorriendo las filas que devuelve la sentencia **select**, se van calculando los pesos y creando los vectores de los documentos, ahorrando bucles y tiempo.

$$w_{ij} = ft_{ij}f_id_j = ft_{ij}\log(d/fd_j) \quad (2.2)$$

## 3. MANUAL DE USUARIO

### 3.1. Instalación

La aplicación se ha desarrollado en Python y su interfaz gráfica de usuario en GTK. Además, ésta requiere de los servicios de un servidor MySQL. Dicho esto, el sistema en el cual se ejecutará la aplicación debe tener instalado el siguiente software:

- Python v2.5 o superior ([5])
- GTK ([1])
- Psycho ([3])
- PyGTK ([4])
- MySQL ([2])
- MySQLdb module for Python ([6])

Inicialmente se debe crear la base de datos que utilizará el sistema documental. En la distribución del software se ofrece un fichero “install” bajo el directorio raíz, que contiene las sentencias necesarias para crear la base de datos y sus tablas mediante el intérprete de MySQL. Para acceder al intérprete de su servidor MySQL escriba la siguiente sentencia en un terminal:

```
mysql -u root -p
```

Introduzca la contraseña que configuró al instalar el servidor de MySQL y cree las tablas con el fichero comentado anteriormente. Llegados a este punto, ya tiene su sistema listo para ser utilizado, pero antes es necesario destacar que éste consiste en dos aplicaciones que pueden ser utilizadas de forma conjunta o por separado: un motor de indexación y un motor de búsqueda. Cada motor posee su propia interfaz gráfica de usuario independiente y, adicionalmente, el motor de indexación posee una interfaz basada en línea de comandos.

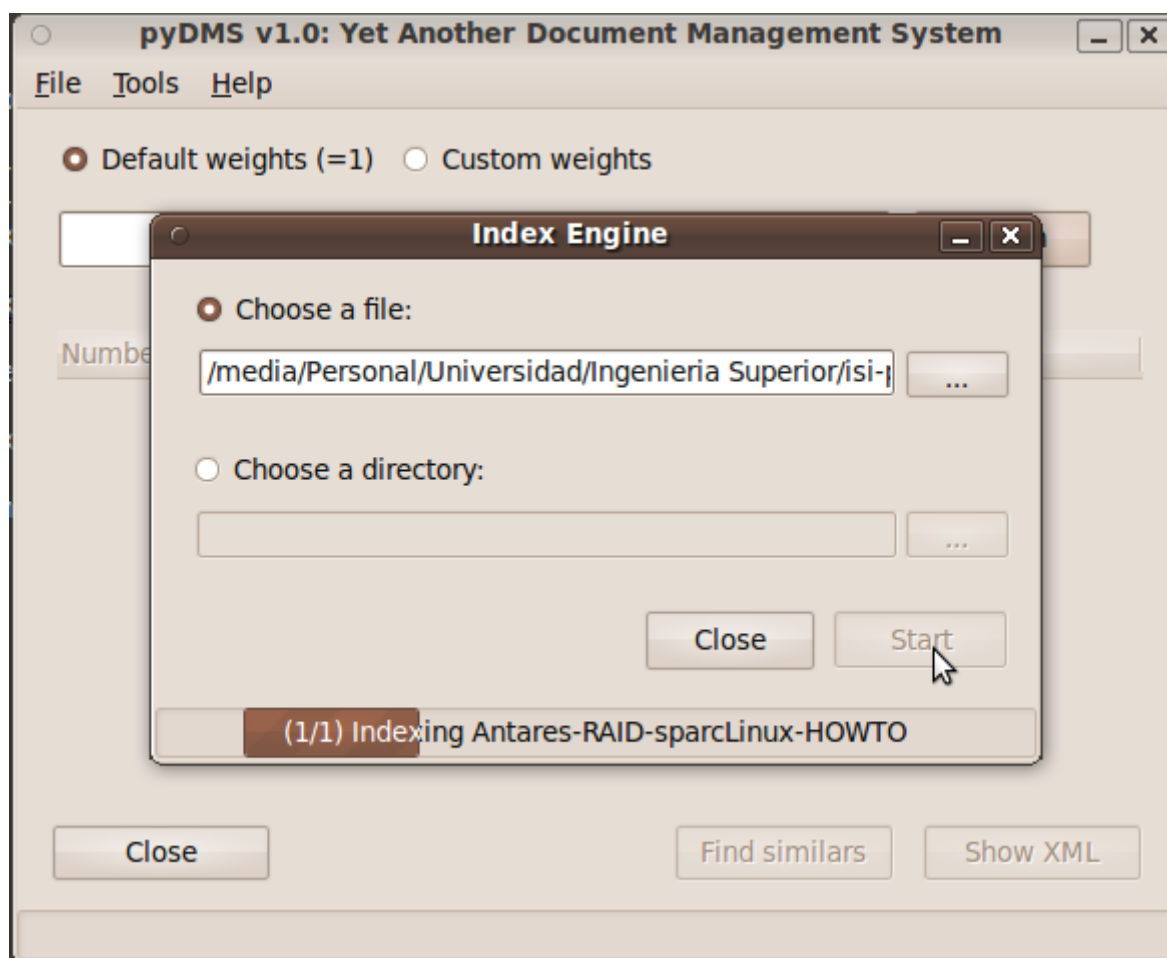
### 3.2. Motor de indexación

#### 3.2.1. Interfaz gráfica de usuario

Para ejecutar la interfaz gráfica sitúese en el directorio “presentación” y ejecute la sentencia `./indexEngineGUI.py`. A continuación se mostrará la ventana principal, tal y como aparece en la Figura 3.1. Las opciones de dicha ventana son:

- **Choose a file:** esta opción sirve para indexar un único fichero. Al pulsar el botón, se abrirá un cuadro de diálogo que permitirá al usuario seleccionar el fichero deseado.
- **Choose a directory:** esta opción sirve para indexar todos los ficheros contenidos en el directorio dado. Al pulsar el botón correspondiente, se abrirá un nuevo diálogo que permitirá al usuario elegir un directorio.

Una vez elegido un fichero o un directorio, basta con hacer clic en el botón “Start” para que comience la indexación, mostrando una barra de progreso para informar al usuario del avance de la indexación de los ficheros.



**Figura 3.1:** Interfaz gráfica de usuario del motor de indexación

### 3.2.2. Interfaz de usuario por línea de comandos

El módulo de indexación además cuenta con una interfaz basada en texto que puede invocarse desde un terminal. Una vez más, esta interfaz está contenida en la capa de presentación del sistema, por lo que hay que introducir la orden (`./indexEngineCMD.py`), siendo necesario proporcionarle dos parámetros, en función de la operación que queramos realizar.

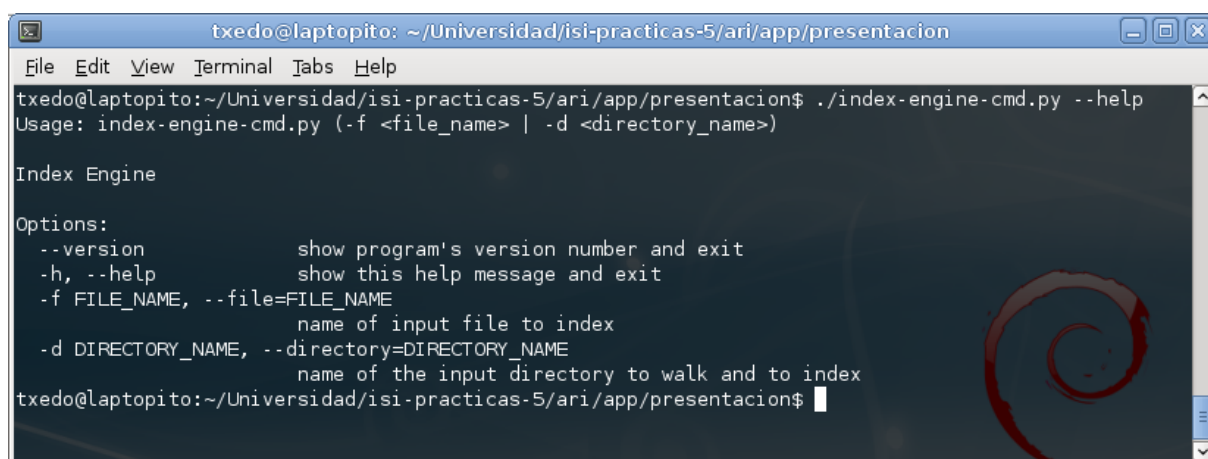
- `[-f | -file] <ruta_a_un_fichero>`. Para indexar un único fichero.
- `[-d | -directory] <ruta_a_un_directorio>`. Para indexar todos los ficheros alojados en un directorio

Si es necesario, se puede consultar la ayuda con el parámetro “-h” o “-help” (ver Figura 3.2)

## 3.3. Motor de búsqueda

Debido a las funcionalidades que debe cubrir esta aplicación, se proporciona una interfaz gráfica de usuario (ver Figura 3.3).

En cuanto a las características de esta aplicación cabe destacar las siguientes características:



```
txedo@laptopito: ~/Universidad/isi-practicas-5/ari/app/presentacion
File Edit View Terminal Tabs Help
txedo@laptopito:~/Universidad/isi-practicas-5/ari/app/presentacion$ ./index-engine-cmd.py --help
Usage: index-engine-cmd.py (-f <file_name> | -d <directory_name>)

Index Engine

Options:
  --version            show program's version number and exit
  -h, --help          show this help message and exit
  -f FILE_NAME, --file=FILE_NAME
                      name of input file to index
  -d DIRECTORY_NAME, --directory=DIRECTORY_NAME
                      name of the input directory to walk and to index
txedo@laptopito:~/Universidad/isi-practicas-5/ari/app/presentacion$
```

**Figura 3.2:** Ayuda para el motor de indexación usando línea de comandos

- Se puede invocar el motor de búsqueda a partir del menú "Tools".
- En el menú "Tools" se pueden consultar los documentos indexados en el sistema (ver Figura 3.4) y ver su contenido, haciendo doble clic en uno de ellos, o pulsando ENTER al seleccionarlo.
- A la hora de realizar una búsqueda podemos configurar los pesos que deseamos dar a cada término. Por defecto, cada término tendrá peso 1, que es el máximo que puede tener, de modo que si queremos quitarle importancia a un término debemos indicarle un peso entre 0 y 1 mediante su barra deslizante correspondiente. Otra característica importante del sistema es que nos permitirá configurar los pesos sobre los términos que realmente formarán parte de la pregunta, ignorando *stopwords* y parseando el texto (ver Figura 3.5).
- Una vez que hemos realizado una búsqueda, se muestran los documentos encontrados ordenados de mayor a menor relevancia con respecto a la pregunta introducida. Además, si hacemos doble clic sobre un documento o lo seleccionamos y presionamos la tecla ENTER, emergerá una ventana mostrándonos su contenido (ver Figura 3.6).
- También podemos buscar documentos similares al documento que tengamos seleccionado. Para ello haremos uso del botón "Find similars".
- Por último, los resultados también pueden ser visualizados en el navegador configurado por defecto en el sistema (ver Figura 3.7). Para ello haremos uso del botón "Show XML".

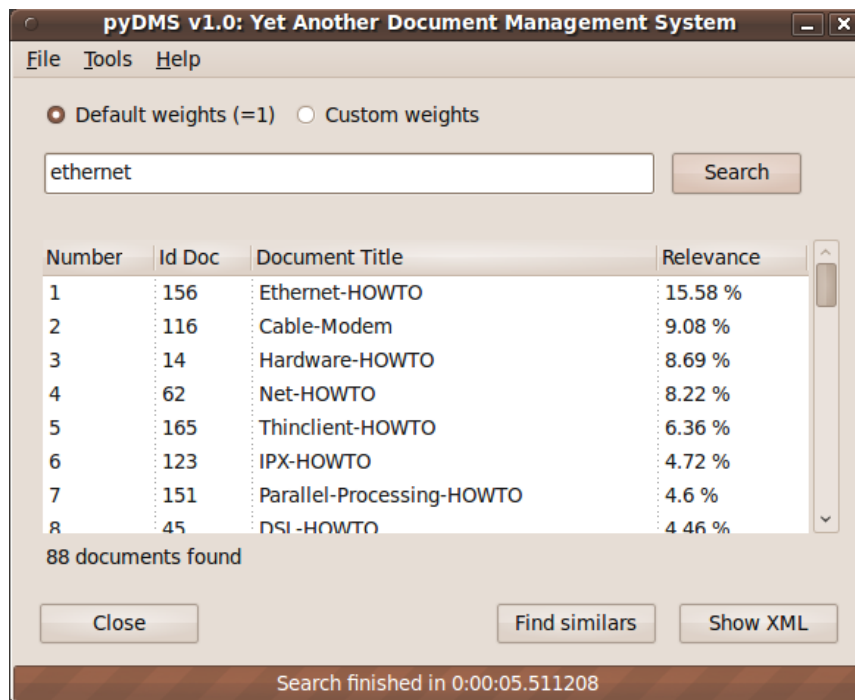


Figura 3.3: Interfaz gráfica de usuario del motor de búsqueda

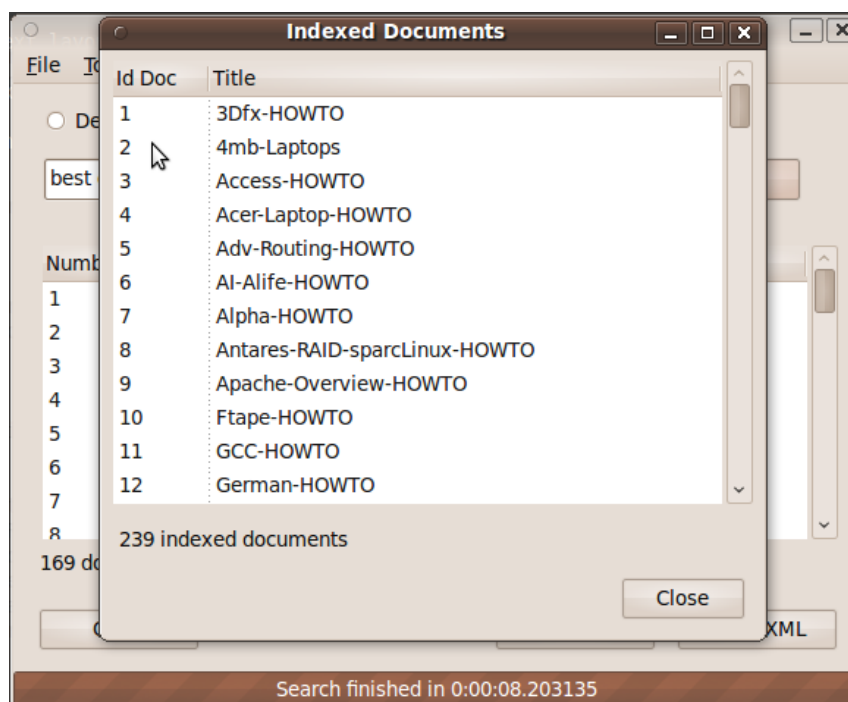
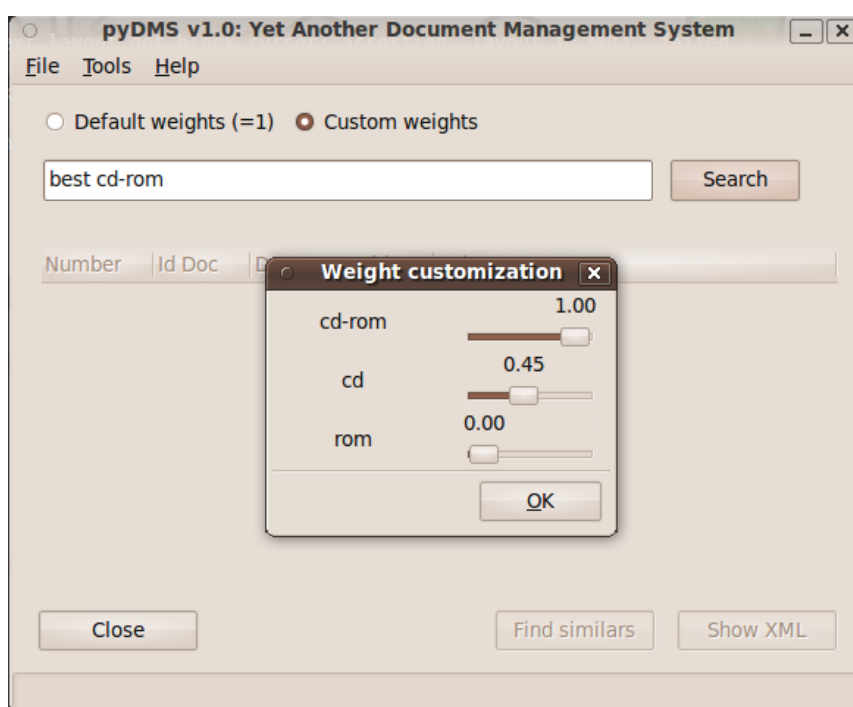


Figura 3.4: Diálogo que muestra los documentos indexados en el sistema



**Figura 3.5:** Configurando los pesos a los términos parseados de una pregunta e ignorando stopwords

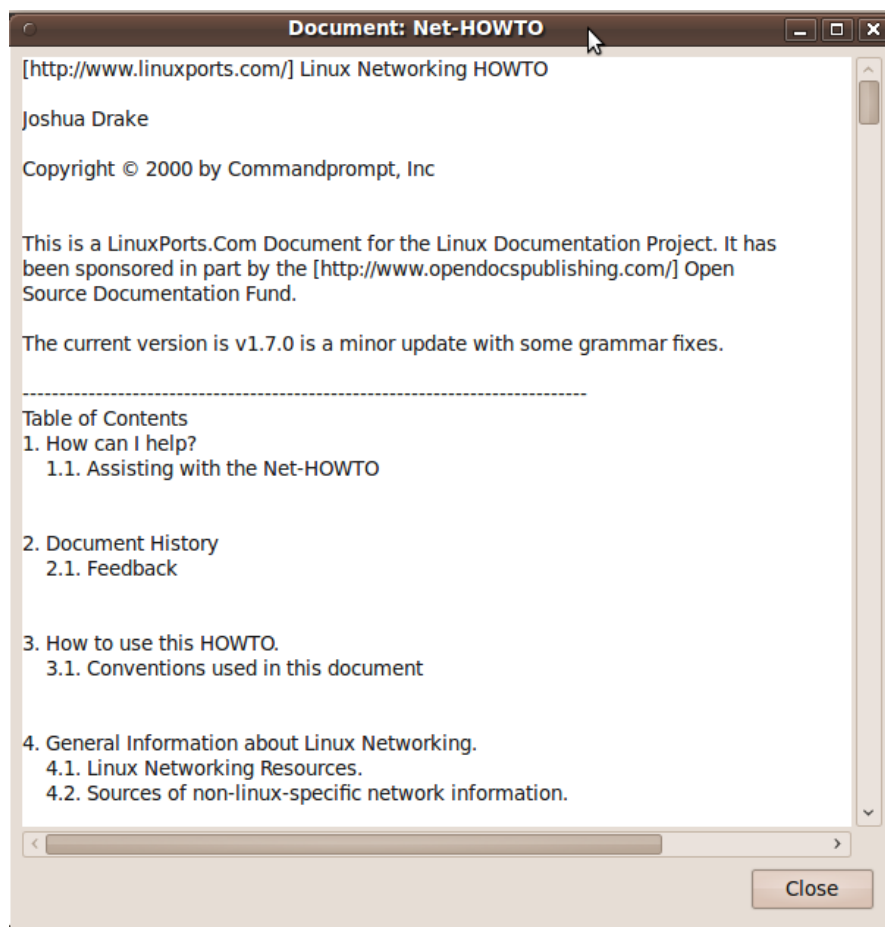


Figura 3.6: Diálogo que muestra el texto del documento seleccionado



Figura 3.7: Resultado de la búsqueda mostrado en el navegador Web



## 4. ESTADÍSTICAS DE FUNCIONAMIENTO

Las pruebas se han realizado en un sistema Ubuntu GNU/Linux 9.10 sobre un ordenador Intel Core 2 Duo@2.5GHz FSB800MHz 4.0GB RAM DRR2@800MHz.

El conjunto de ficheros de muestra se compone de un total de 239 documentos.

### 4.1. Motor de indexación

En el primer motor de indexación que se desarrolló, el tiempo que se obtuvo para indexar los 239 documentos fue de **27 minutos y 39 segundos**.

Con el cambio en el diseño de la base de datos (ver sección 2.3) y la optimización de las sentencias **insert** en la tabla *dic* y en la tabla *positng\_file*, el tiempo que se ha obtenido es de **4 minutos y 27 segundos**. Como se puede observar, la disminución en el tiempo es bastante considerable.

Por otra parte, al igual que ocurría en la primera versión de este motor, la memoria caché no disminuye el número de acceso a disco. Esto es debido a que la caché (configurada por defecto a 3000 líneas) no llega a su máxima capacidad, ya que debe vaciarse tras analizar cada documento, para mantener la integridad de clave ajena entre la tabla *positng\_file* y la tabla *dic*. Si los documentos tuviesen más términos, la caché llegaría a ser efectiva.

### 4.2. Motor de búsqueda

En la búsqueda, como se comentó en la sección 2.3, se ha optado por utilizar la última sentencia **select**, apoyándose en la creación de la vista temporal. Se ha elegido esta opción, porque, de media, tarda unos 5 segundos en buscar el término "linux" (que aparece en la mayoría de los documentos), mientras que con la sentencia **select** que incluía una subconsulta, buscar el mismo término tardaba más de 40 segundos.

En la tabla 4.1 se muestra un resumen de búsquedas que se han realizado en el sistema (los tiempos son una media, ya que el tiempo de búsqueda puede variar según la carga del procesador y la asignación de éste a los hilos). Como se puede observar, el tiempo varía según la cantidad de términos introducidos y el número de documentos en los que aparece ese término.

Pregunta	Tiempo empleado	Documentos encontrados
linux	4.8 segundos	237 documentos
LoopBack	2.5 segundos	33 documentos
0.0.0.25	0.22 segundos	1 documento
127.0.0.1	2.1 segundos	29 documentos
ETHERNET 127.0.0.1 NETWORK	3.6 segundos	177 documentos
cd-rom	4.9 segundos	167 documentos
Similar a "Installation- HOWTO"	5.5 segundos	239 documentos
Similar a "CDROM- HOWTO"	5.7 segundos	239 documentos
Similar a "Assembly- HOWTO"	6.1 segundos	239 documentos

**Tabla 4.1:** Tabla resumen de los resultados de búsquedas

## REFERENCIAS

- [1] The GTK+ Project. <http://www.gtk.org/>.
- [2] MySQL - Official Website. <http://www.mysql.com/>.
- [3] Psyco Home Page. <http://psyco.sourceforge.net/>.
- [4] Pygtk: GTK+ Project for Python. <http://www.pygtk.org>.
- [5] Python Programming Language - Official Website. <http://www.python.org/>.
- [6] MySQLdb: MySQL module for Python. <http://sourceforge.net/projects/mysql-python/>.