UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

Almacenamiento y Recuperación de la Información

Módulo de importación e indexación de documentos

21 de octubre de 2009

Juan Andrada Romero
Jose Domingo López López

Índice

1. Introducción

Este documento aborda uno de los módulos del sistema a desarrollar, el cúal es el módulo de importación e indexación de documentos. En la sección 2 se tratarán las decisiones de diseño tomadas a la hora de desarrollar este módulo. En la sección 3 se explica brevemente el funcionamiento de esta aplicación, en la sección 4 se comentan algunas estadísticas de funcionamiento del sistema y, para terminar, en la sección 5 se proponen algunas mejoras para desarrollar en futuras versiones del sistema.

2. Decisiones de diseño

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 de este módulo del sistema, ya que Python los gestiona de una manera eficaz y permite su uso de uso de una manera sencilla.

Por otra parte, hemos elegido un sistema UNIX porque nos resulta más cómodo a la hora de implementar el sistema.

2.2. Diseño de la base de datos

Para implementar los elementos necesarios para la indexación de documentos, como son el *Posting_File*, la tabla de documentos y el diccionario de términos, se ha optado por utilizar una base de datos relacional, siguiendo el diagrama mostrado en la Figura 2.1. Las tablas que se representan 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.
- **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 *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.

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.

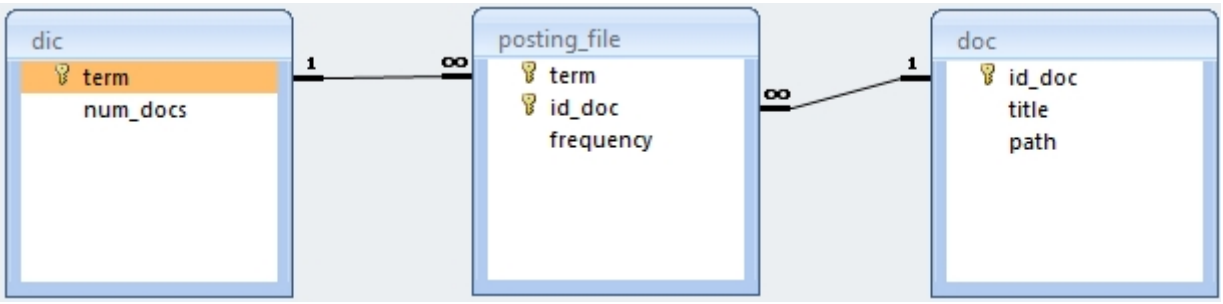

Figura 2.1: Diagrama de la base de datos

2.3. Desarrollo del sistema

Para terminar, en esta subsección se comentan las diferentes decisiones de diseño, así como los problemas que han surgido a la hora de desarrollar el módulo de importación e indexación de documentos.

2.3.1. Diseño multicapa

Este módulo 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.

Un ejemplo de la extensibilidad del sistema es que en la capa de presentación se han implementado dos interfaces de usuario, una en modo gráfico y otra por línea de comandos, tal y como se detallan en la sección 3.

En la capa de dominio, se almacenan las clases que implementan esta aplicación, como son el analizador de los documentos y las clases auxiliares que utiliza, como las cachés y un archivo de configuración. Las clases de la capa de presentación conocen a las clases de dominio (en concreto, al analizador), pero no al revés, por lo que podría cambiarse la interfaz de usuario sin tener que cambiar las clases de dominio.

Por otra parte, en la capa de persistencia se encuentran las clases que se encargan de almacenar la información en la base de datos. Dichas clases implementan un patrón Agente Singleton y un patrón fabricación pura o DAO. El primer patrón es el que se encarga 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.

Por su parte, el patrón DAO se ha utilizado para desacoplar el dominio de la persistencia, evitando así que las propias clases del dominio creen las secuencias 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.

2.3.2. Analizador de documentos

Conexión con la base de datos En una primera iteración del desarrollo de la aplicación, 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 la aplicación, y cerrarla cuando la aplicación finaliza.

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 método *parser*. Dicho módulo recibe una línea del documento a indexar y realiza la siguientes acciones:

- En primer lugar, se escapan los caracteres "\" y " ' " de las palabras que los contengan, ya que si se intenta insertar un término con esos caracteres, 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.
- Se sustituyen las vocales acentuadas de las palabras por vocales sin acentuar.
- En cada una de las palabras de la línea, se sustituyen los separadores que pueda contener la palabra por un espacio en blanco y se divide la palabra por dichos espacios, obteniendo así una lista de palabras, que recibe el siguiente tratamiento:
  1. Se eliminan los espacios en blanco.
  2. Si alguna palabra de la lista que se ha obtenido al dividir una palabra por separadores se encuentra en la stop_list, dicha palabra no se divide, ya que quedarían palabras sin sentido. Por ejemplo, si la palabra "F-14" se divide en "[F," ",14]", al encontrarse "F" en la stop_list, se almacenaría todo el término sin separarse.
  3. Si al dividir la palabra solo se obtiene una palabra de la stop_list rodeada de espacios, dicha palabra se ignora. Por ejemplo, al separar la palabra "\t already?", se obtendria la lista [" ", " ", already," "], que sólo contiene una palabra de la stop_list rodeada de espacios, lo cual no tiene sentido.
  4. En las direcciones web o e-mail, el tratamiento es el mismo: se divide la palabra por separadores, obteniendo una lista y almacenando las palabras por separado.
     Esto soluciona el siguiente problema: si el texto no tiene una escritura correcta y apareciese la palabra "juanro.1987@gmail.com.Hola", con este tratamiento se almacenarían las palabras "juanro", "1987", "gmail", "com" y "hola". Sin embargo, si tratasemos un e-mail como una palabra completa hasta que se encontrase un espacio, se almacenaría como término "juanro.1987@gmail.com.Hola", que luego no se encontraría en el diccionario si alguien consulta por la dirección de e-mail "juanro.1987@gmail.com". En nuestro caso, como este mismo parser se va a aplicar a las consultas, la cadena anterior recibiría el mismo tratamiento y podría recuperar los términos que se habían almacenado en el diccionario por separado.

En resumen, este módulo parser se ha implementado para dar el mismo tratamiento tanto a las cadenas de los documentos a indexar como a las cadenas de consultas.

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 corrrecto 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.

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 diccionario a la base de datos cuando se termina de procesar el documento, en lugar de ir accediendo a la base de datos por cada uno de los términos que se deben almacenar.

Memoria caché Se ha implementado una memoria caché, pero, tal como se puede analizar en la sección de estadísticas (4), no se han obtenido los resultados esperados (minimizar el número de accesos a disco). La idea se ha implementado con dos diccionarios:

- `new`. Contiene nuevos términos que no están en la base de datos y que serán insertados con la sentencia "INSERT".
- `old`. Contiene términos que ya estaban en la base de datos, es decir, que han sido recuperados de ésta, y serán actualizados con la sentencia "UPDATE".

Dado que no hemos llegado al suficiente grado de granuralidad en este diseño, aunque no minimizamos el número de accesos a disco, al menos hacemos que éstos sean en intervalos más espaciados en el tiempo, de modo que no se ocupa el recurso de forma constante.

Psyco Psyco 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.

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])
- Psyco ([3])
- PyGTK ([4])
- MySQL ([2])
- MySQLdb module for Python ([6])

Ahora 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 "persistencia", que contiene las sentencias necesarias para crear la base de datos y sus tablas mediante el intérprete de MySQL. Si no sabe acceder al intérprete de su servidor MySQL escriba la siguiente sentencia en un terminal:

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

Introduzca la contraseña que estableció 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.

A continuación se hablará de cada una de las interfaces disponibles en el sistema para el usuario.

3.2. Interfaz gráfica de usuario

Al ejecutar la interfaz gráfica (sitúese en el directorio "presentación" y ejecute la sentencia `./index-engine-gui.py`) 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 llamado "Start" para que comienze la indexación, mostrando una barra de progreso para informar al usuario del avance de la indexación de los ficheros.

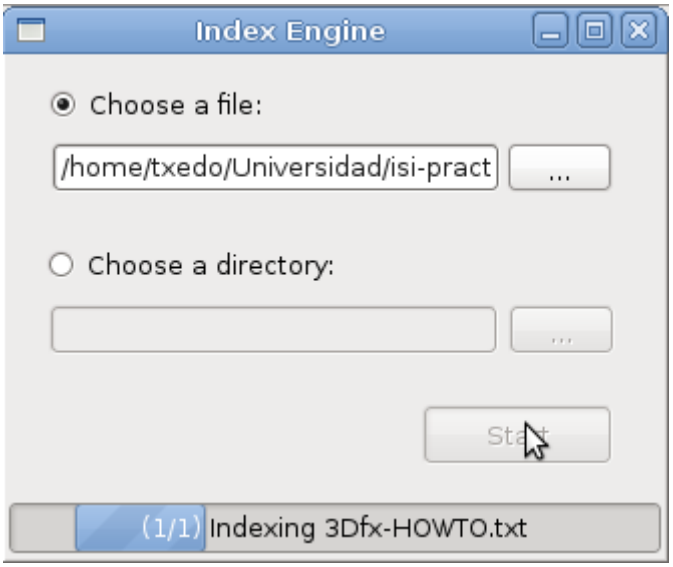

Figura 3.1: Interfaz de usuario gráfica

3.3. Interfaz de usuario por línea de comandos

La aplicación 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. Para lanzarla, es 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)

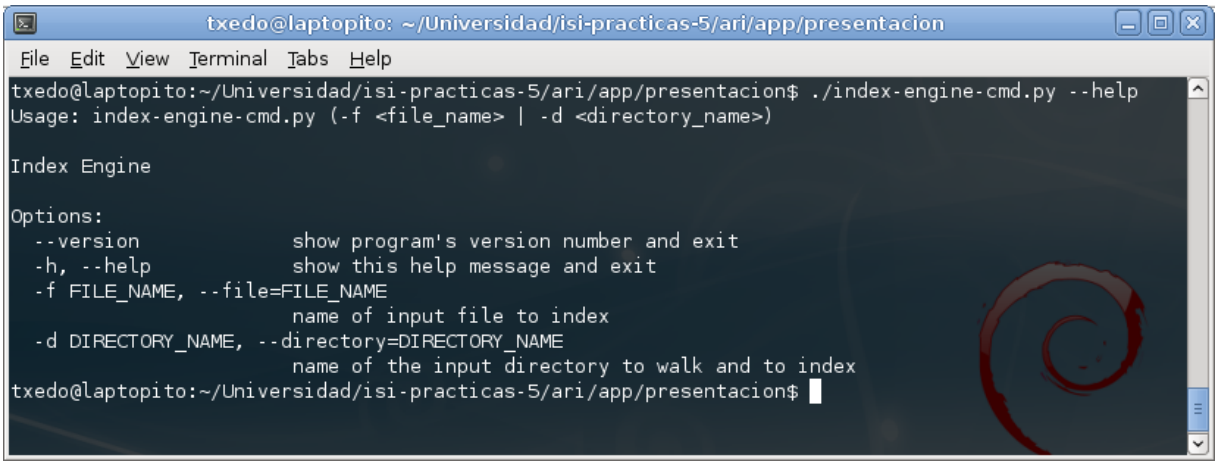

Figura 3.2: Interfaz de usuario gráfica

4. Estadísticas de funcionamiento

Las pruebas se han realizado en un sistema Debian GNU/Linux 4.0 Etch sobre un ordenador Intel Centrino@2.0GHz FSB800MHz 2.0GB RAM DRR2@800MHz.

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

Ya sea utilizando una memoria caché de tamaño 1 (caso base, comportamiento similar al de un sistema sin caché) como de tamaño 1000, el tiempo que se ha obtenido en la indexación es similar: entorno a los 27 minutos. La diferencia radica en que en el primer caso el acceso a disco y el uso del procesador se mantiene constante, mientras que en el segundo caso, ambos usos de recursos vienen dados por intevalos, coincidiendo éstos con la sincronización de la caché con la base de datos.

No obstante, estos resultados nos hacen meditar en que el uso de la memoria caché no ha mejorado la eficiencia del sistema tanto como se esperaba, por lo que en próximas versiones se harán las revisiones pertinentes.

5. Futuras mejoras

Cache Como se dice en la sección de estadísticas (sección 4), la memoria caché no es tan óptima como se esperaba. En próximas versiones se tratará de afinar la granuralidad de esta funcionalidad para así minimizar, en la medida de lo posible, el número de accesos a disco.

Extensibilidad El sistema es completamente extensible gracias a su arquitectura multicapa. Como prueba de ello se han incorporado dos interfaces de usuario, una gráfica y otra basada en texto, pero se quiere explotar más las posibilidades que nos brinda esta arquitectura. Para ello se tratará de preparar la capa de persistencia de modo que haya modos de almacenamiento alternativos a la base de datos relacional que se utiliza actualmente.

Uso de disco Aunque esta mejora ya está implícita en el uso de memorias caché, se tratará de mejorar esta característica en otro ámbito. Actualmente, el sistema lee línea a línea el fichero que está parseando para su indexación, por lo que se están solicitando accesos a disco de forma continua (uno por cada línea). Para reducir esta operación a un sólo acceso a disco se leerá el fichero completo y se guardará en la memoria RAM, de modo que el acceso a cada una de las líneas será más eficiente. Esto es posible ya que los ficheros con los que trabaja el sistema son de texto plano y no tienen un gran volumen.

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/.