2013

Proyecto Final

ORGANIZACIÓN DE ARCHIVOS

ARIEL ESQUIVEL Y HENRY ACOSTA

UNITEC | Ingeniería en Sistemas Computacionales

Introducción

El presente informe tiene como propósito presentar el proceso de desarrollo del trabajo asignado como proyecto final de la clase de Organización De Archivos, describiendo cada una de las funciones que lo componen. El proyecto consta de la realización de un programa que gestione archivos de registros de manera eficiente evitando acceder demasiadas veces al disco duro, para esto utilizamos las técnicas aprendidas en clase.

Marco Teórico

¿Qué es un registro?

Es un conjunto de información compuesta por campos.

¿Qué es un campo?

Es un espacio donde se almacena un dato.

El disco duro es bastante lento con respecto a la memoria principal y en este proyecto nuestro principal objetivo es acceder lo menos posible al disco duro, para lograr esto existen diferentes técnicas para organizar registros en un archivo, ejemplos:

- Registros de longitud fija
- Registros con un número fijo de campos
- Utilizar un indicador de longitud al inicio de cada registro
- Utilizar un delimitador para separar los registros.
- Hacer uso de un segundo archivo con la información necesaria para acceder directamente a un registro.

Nosotros utilizamos un archivo de índices que guarda la llave, la posición y la longitud de cada registro. Con esa información somos capaces de acceder de forma rápida a un registro en particular.

Para entender mejor lo que significa un archivo de índice, vamos a definirlo.

Archivo de índice: Estructura auxiliar diseñada para optimizar la velocidad de acceso a registros.

Para no trabajar directamente con el archivo de índice, existen estructuras de datos que nos permiten tener esa información en memoria principal y de esta manera realizar más rápido las operaciones. Estructuras:

- Árboles binarios de búsqueda
- Árboles AVL
- Árboles binarios de búsqueda paginados
- Árbol B
- Árbol B+

Para nuestro programa hicimos uso de un árbol b. La función de búsqueda de un registro en nuestro programa se hace por medio de una llave y está disponible realizar la búsqueda utilizando un índice simple o un árbol b, de esta forma se puede hacer una comparación de qué estructura es más rápida.

Implementación

El programa fue desarrollado utilizando el lenguaje C++ y las bibliotecas de Qt para la interfaz gráfica. Se puede descargar Qt desde la siguiente dirección: http://qt-project.org/downloads

El proyecto consta de las siguientes clases:

1- RecordOperations: esta clase nos permite realizar varias operaciones sobre el archivo de registro, las funciones son las siguientes:
   - int getNumberOfFields(): retorna la cantidad de campos que contiene el archivo de registro.

   - int getLengthOfTheNumberOfFields(): retorna la cantidad de caracteres que posee el “número de campos”.

   - int getSizeOfAFieldInformation(int): retorna el tamaño de la información de un campo localizado en la posición especificada del archivo de registro.

   - int getLengthOfTheSizeOfAFieldInformation(int): retorna la cantidad de caracteres del “tamaño de la información de un campo” localizado en la posición especificada del archivo de registro.

   - int getLengthOfFieldsInformation(): retorna el tamaño de toda la información de los campos.

- QStringList getFieldsInformation(): retorna la información de los campos como una lista separada por comas.

- int getInitialPositionOfRecordsInformation(): retorna la posición del archivo de registro justo después del carácter ":" que sirve como separación entre la información de los campos y los registros.

- int getNumberOfRecords(): retorna la cantidad de registros que contiene el archivo de registros.

- int getLengthOfTheNumberOfRecords(): retorna la cantidad de caracteres que contiene el "número de registros".

- int getSizeOfARecordInformation(int): retorna el tamaño de un registro localizado en la posición especificada del archivo.

- int getLengthOfTheSizeOfARecordInformation(int): retorna la cantidad de caracteres que posee el "tamaño de un registro" localizado en la posición especificada del archivo.

- int getLengthOfRecordsInformation(): retorna la longitud de todos los registros.

- int getRecordPositionAt(int): retorna la posición de un registro en el archivo localizado en el índice especificado.

- QStringList getRecordInformationAt(int): retorna el registro localizado en el índice especificado del archivo.

- QStringList getRecordsInformation(): retorna todos los registros separados por una comma.

2- CreateFieldDialog: esta clase es la encargada de la creación de un nuevo campo, las funciones son las siguientes:
   - bool fieldAlreadyExist(): verifica si el campo que se desea ingresar ya existe en el archivo de registros.

   - bool keyAlreadyExist(): verifica si existe un campo llave en el archivo de registro.

   - bool addField(): agrega el campo y su información en el archivo de registro.

   - void on_comboBoxType_currentIndexChanged(int index): decide si el usuario puede establecer un valor para la cantidad de decimales de la longitud del campo en base al tipo de dato del combo box.

   - void on_checkBoxKey_clicked(): si el usuario decide que el campo debe ser llave y marca el check box, automáticamente se estable el tipo de información como un número y no se puede cambiar eso.

   - void on_pushButtonAccept_clicked(): guarda el campo.

   - void on_pushButtonCancel_clicked(): cierra la ventana.

3- ModifyFieldDialog: se encarga de modificar la información de los campos, contiene las siguientes funciones:
   - void tableProperties(): establece algunas propiedades de la tabla.

   - void showFields(): Muestre la información de los campos en la tabla.

   - void modifyName(): modifica el nombre del campo.

   - void modifyTypeLengthDecimal(): modifica el tipo, la longitud y la cantidad de decimales que puede tener el campo.

   - void modifyKey(): modifica la llave.

   - void modifyField(int, int, QString): función principal para modificar alguna información del campo, es la que internamente llaman las otras funciones.

4- MainWindow: es la clase principal, desde aquí se realizan todas las operaciones, consta de las siguientes funciones:
   - void init(): realiza algunas operaciones iniciales para que todo funciones bien.

   - bool validFile(): verifica si el archivo que se abre es compatible con el programa.

- bool hasIndexFile(): verifica si el archivo de registros posee su respectivo archivo de índice.

- bool isIndexFileCorrupted(): verifica si el archivo de índice esta corrupto, esto es cuando la lista de índices posee menos índices que la cantidad de registros del archivo de registros.

- void loadIndexFile(): carga la información del archivo de índice y lo guarda en la lista de índices.

- void showFields(): muestra los campos en la tabla de registros.

- void showRecords(): muestra los registros en la tabla de registros.

- bool insertRecord(): guarda un registro, dependiendo si el usuario ha borrado un registro anteriormente o el availList está vacío.

- bool insertRecordEOF(): guarda el registro al final.

- bool insertRecordAvailList(): guarda el registro en alguna posición almacenada en el availList donde quepa.

- bool isValidItem(int, int): verifica si el tipo de dato que se quiere almacenar sea el correcto.

- bool isValidItemLength(int, int): verifica si la longitud del dato que se quiere almacenar contenga una longitud valida.

- bool isKey(int): verifica si la columna especificada es llave.

- bool isValidKey(int, int): verifica si el dato que se va a almacenar no este repetido, ya que es llave.

- bool indexListSearch(QString): busca una llave en el archivo de registros utilizando la lista de índice.

- bool bTreeSearch(QString): busca una llave en el archivo de registros utilizando el árbol-b.

- bool compact(): compacta el archivo de índices, esto es borrar los espacios de los registros eliminados.

- bool saveIndexList(): almacena lo que contiene la lista de índice en el archivo de índice.

- void on_actionNewFile_triggered(): crea un nuevo archivo de registro con su respectivo archivo de índice.

- void on_actionOpenFile_triggered(): abre un archivo de registro creado anteriormente.

- void on_actionSaveFile_triggered(): guardo los cambios hechos en la tabla. Esto implica compactar, actualizar el archivo de índice, borrar, agregar y en algunos casos recrear el archivo de índice.

- void on_actionPrintFile_triggered(): guarda los registros en un pdf.

- void on_actionCloseFile_triggered(): cierra el archivo con el que se esta trabajando.

- void on_actionExit_triggered(): salir del programa.

- void on_actionCreateField_triggered(): muestra la ventana para crear un campo.

- void on_actionModifyField_triggered(): muestra la ventana para modificar un campo.

- void on_actionCrossRecords_triggered(): cruza un archivo con otro.

- void on_actionCreateSimpleIndex_triggered(): crea el archivo de índice en base al archivo de registro. Esto se va a poder realizar cuando al abrir un archivo de registro se detecte que no existe el archivo de índice correspondiente.

- void on_actionCreateBTreeIndex_triggered(): crea el índice en memoria utilizando un árbol-b. Este índice se va a utilizar para realizar una búsqueda.

- void on_actionReindexing_triggered(): re indexa el archivo de índice. Esto se va a poder realizar cuando se abra un archivo de registro y se detecte que la lista de índice

contiene menos elementos que la cantidad de registros del archivo de registros, entonces se actualiza el archivo de índice y se agrega la información de los registros que faltan.

- void on_actionImportXML_triggered(): importa un archivo XML que se haya exportado desde el programa anteriormente.

- void on_actionExportXML_triggered(): exporta el archivo de registro a formato XML.

- void on_actionImportJSON_triggered(): importa un archivo JSON que se haya exportado desde el programa anteriormente.

- void on_actionExportJSON_triggered(): exporta el archivo de registros a formato JSON.

- void on_actionAbout_triggered(): muestre información acerca del programa.

- void on_tableWidgetRecords_customContextMenuRequested(const QPoint &pos): muestra un menú emergente al presionar el botón derecho del mouse que contiene la opción de agregar una fila y borrar un registro.

- void insertRow(): inserta una fila en la tabla de registros.

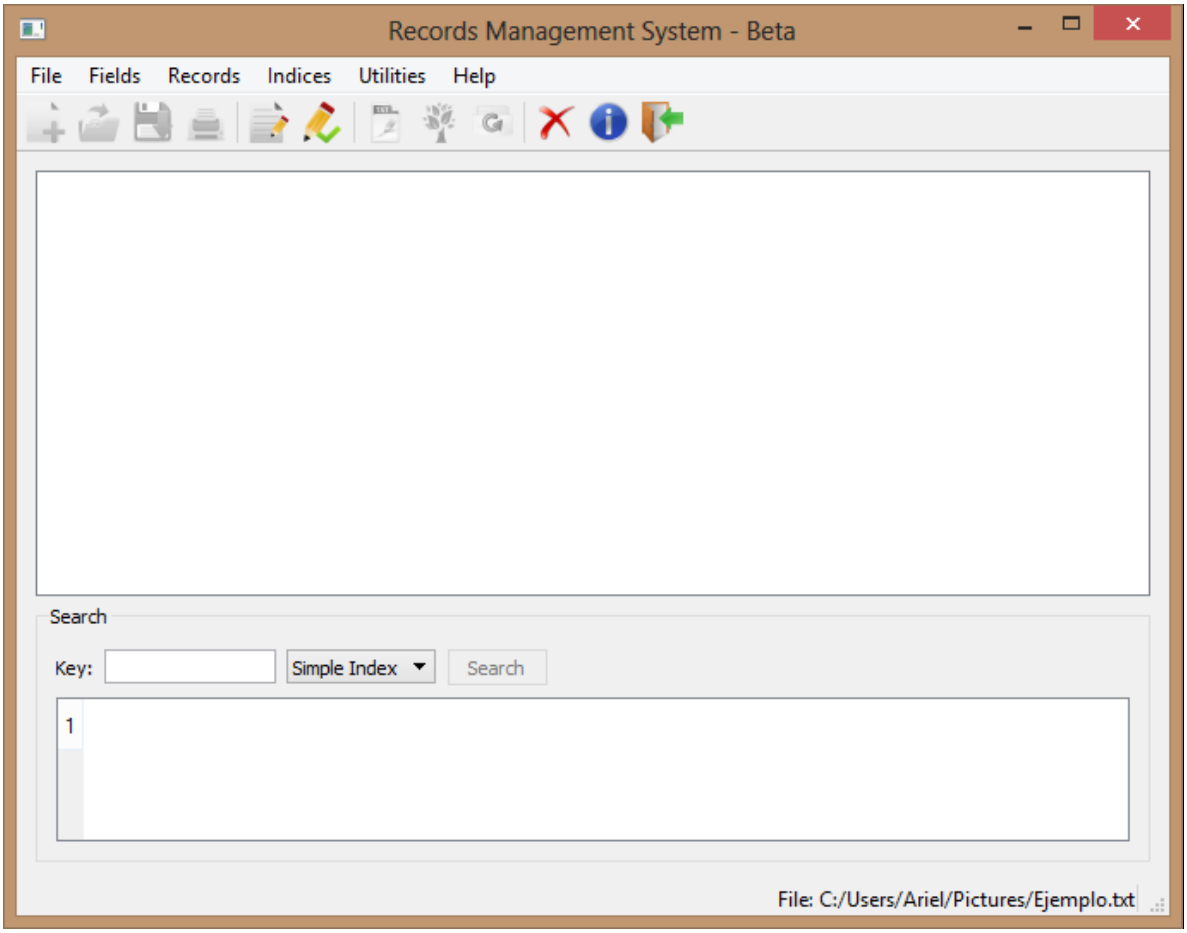
- void deleteRecord(): elimina un registro.

- void on_tableWidgetRecords_cellChanged(int row, int column): valida el registro que se quiere ingresar y lo guarda.

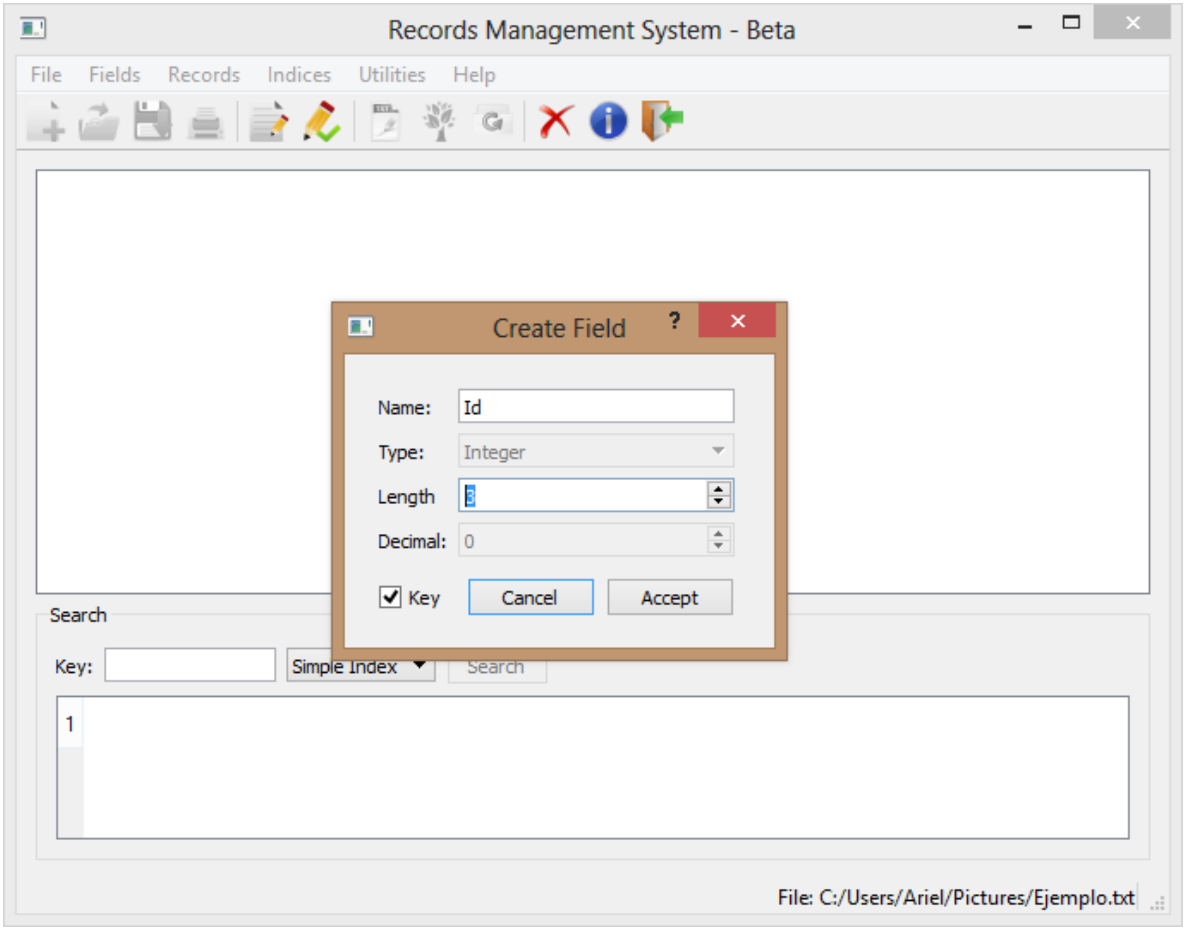
- void on_pushButtonSearch_clicked(): realiza una búsqueda utilizando la lista de índice o el árbol-b dependiendo de la opción que el usuario elija.

Ejemplo

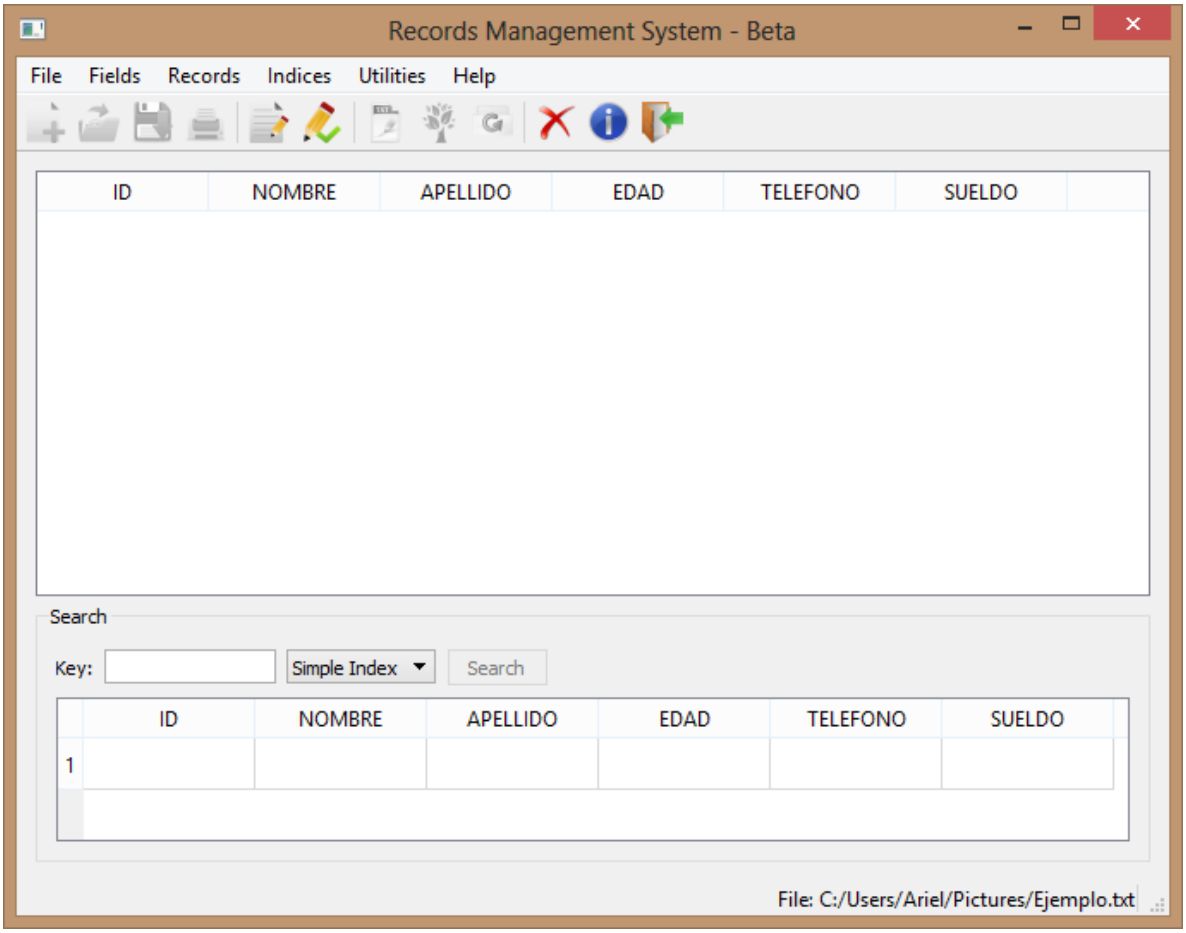
Ventana principal:



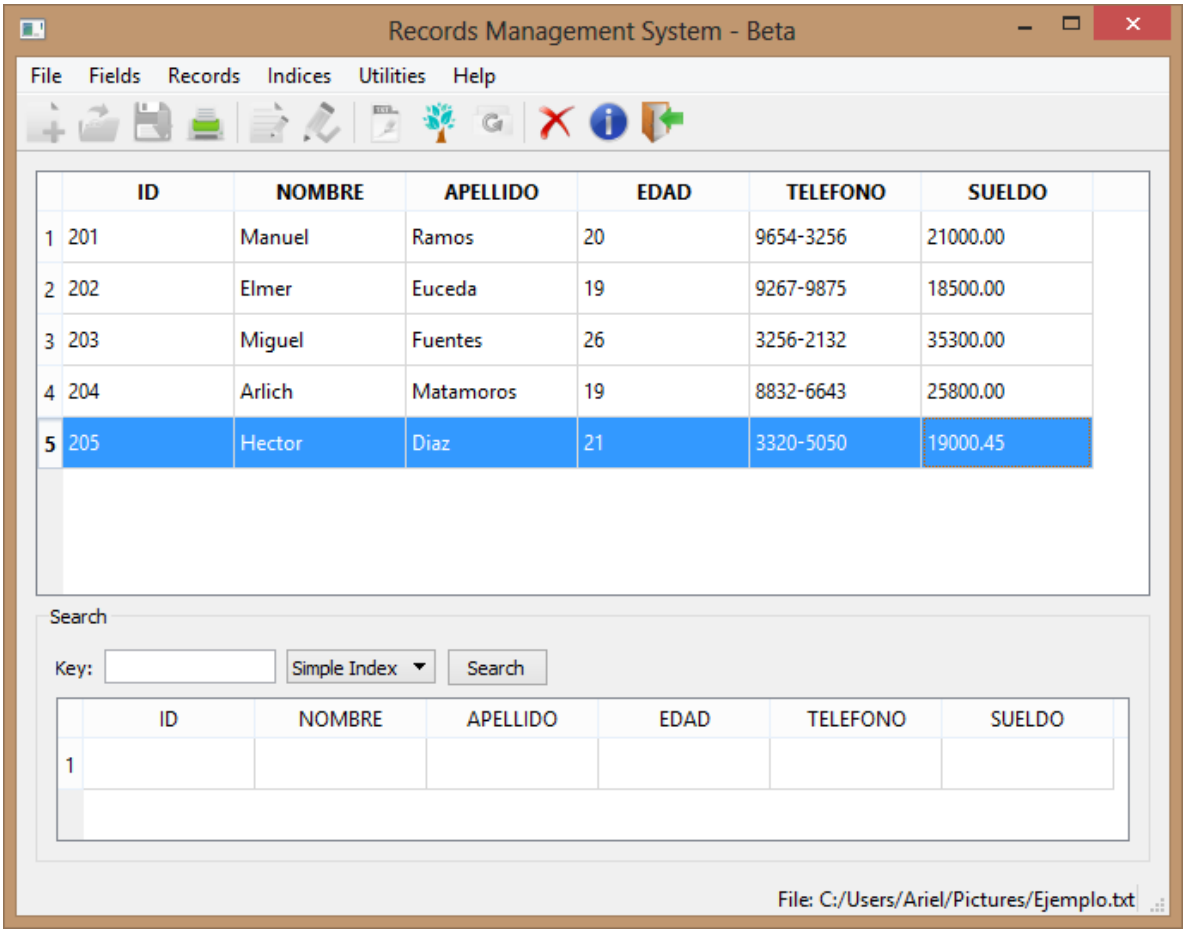
Agregando campos:



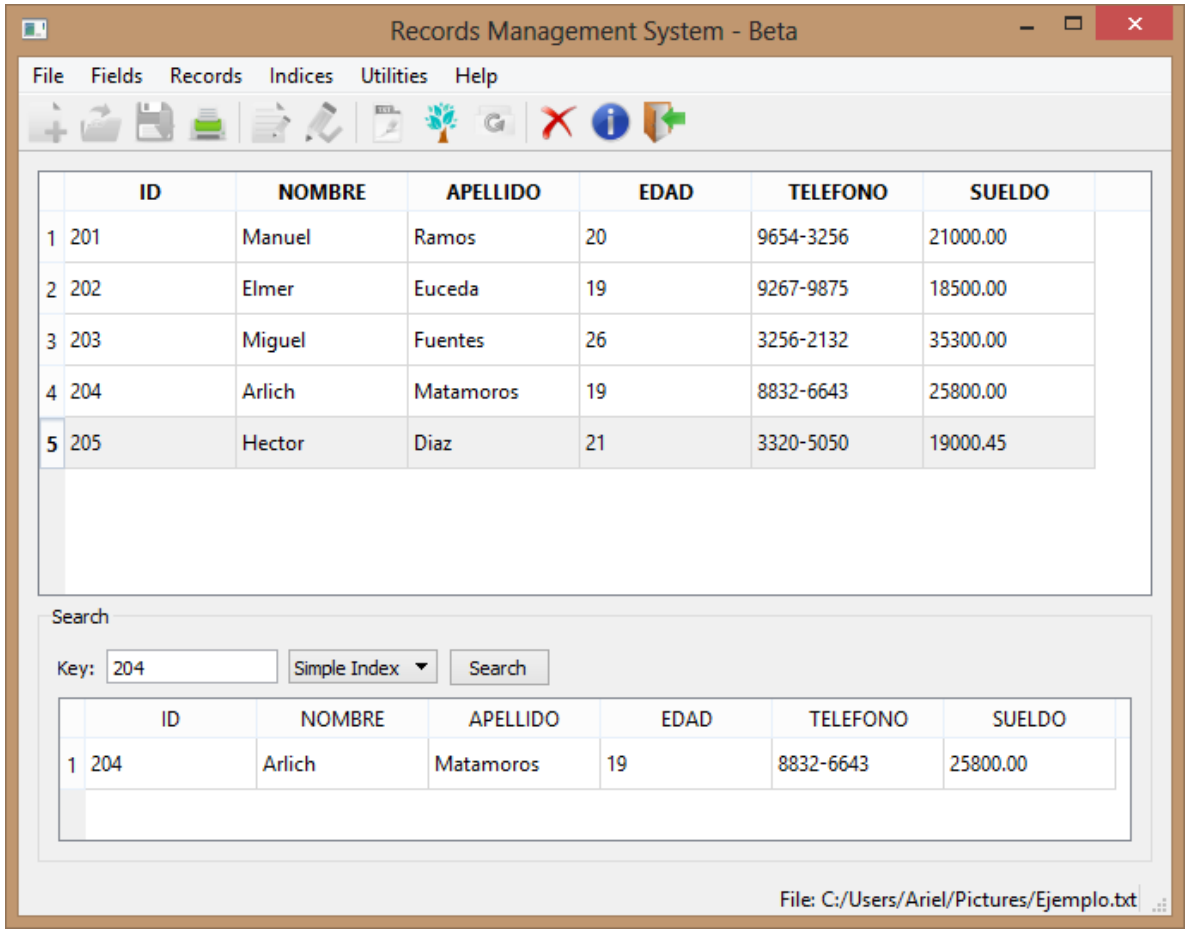
Ventana principal con campos:



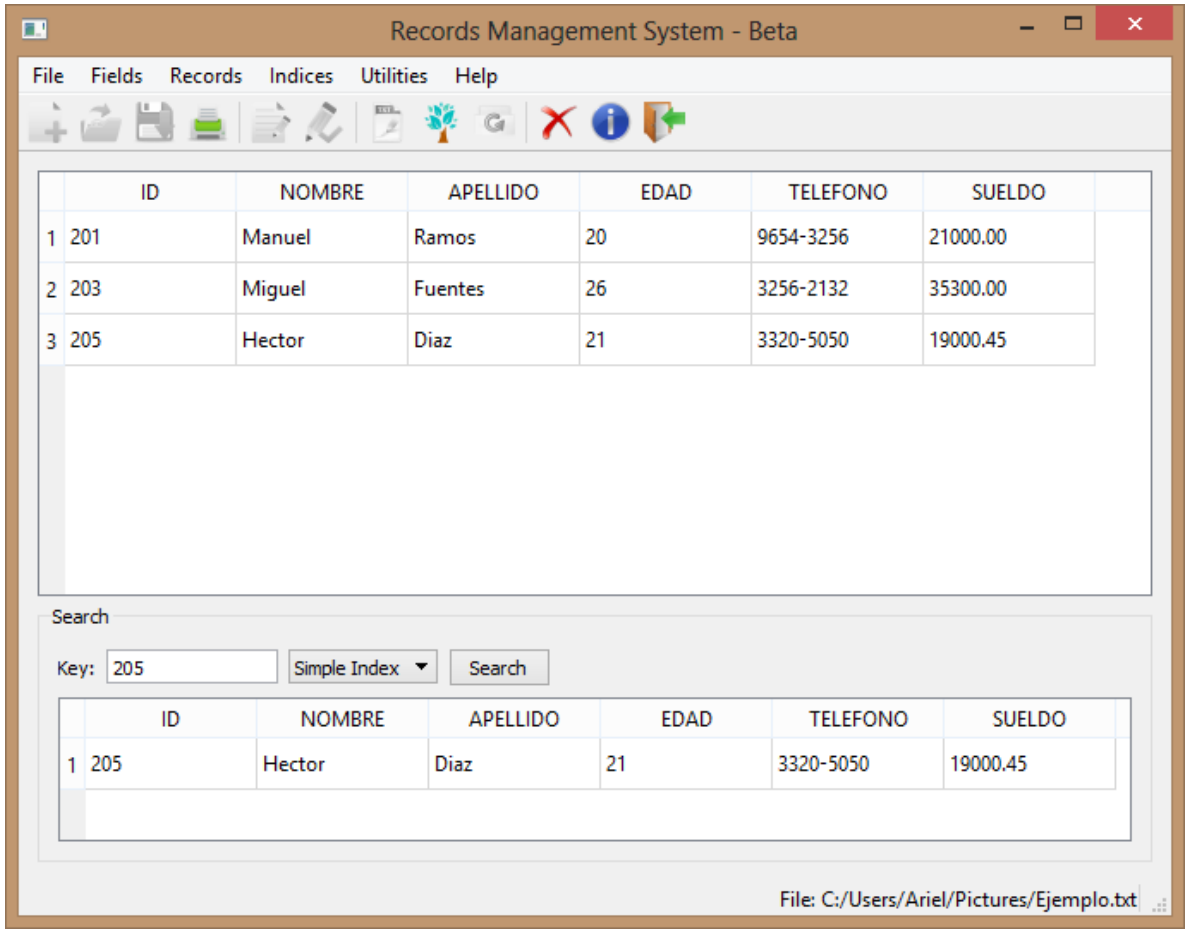
Agregando registros:



Buscando:


Records Management System - Beta
File
Fields
Records
Indices
Utilities
Help
ID
NOMBRE
APELLIDO
EDAD
TELEFONO
SUELDO
1 201 Manuel Ramos 20 9654-3256 21000.00
2 202 Elmer Euceda 19 9267-9875 18500.00
3 203 Miguel Fuentes 26 3256-2132 35300.00
4 204 Arlich Matamoros 19 8832-6643 25800.00
5 205 Hector Diaz 21 3320-5050 19000.45
Search
Key: 204
Simple Index
Search
ID
NOMBRE
APELLIDO
EDAD
TELEFONO
SUELDO
1 204 Arlich Matamoros 19 8832-6643 25800.00
File: C:/Users/Ariel/Pictures/Ejemplo.txt

Eliminando:


Records Management System - Beta
File
Fields
Records
Indices
Utilities
Help
ID
NOMBRE
APELLIDO
EDAD
TELEFONO
SUELDO
1 201 Manuel Ramos 20 9654-3256 21000.00
2 203 Miguel Fuentes 26 3256-2132 35300.00
3 205 Hector Diaz 21 3320-5050 19000.45
Search
Key: 205
Simple Index
Search
ID
NOMBRE
APELLIDO
EDAD
TELEFONO
SUELDO
1 205 Hector Diaz 21 3320-5050 19000.45
File: C:/Users/Ariel/Pictures/Ejemplo.txt

Conclusiones

Luego de concluir el desarrollo de la aplicación y realizar diversas pruebas, podemos llegar a la conclusión que aplicando las técnicas correctas para trabajar con la lectura/escritura en un disco duro, de tal manera que el uso del mismo sea mínimo, se puede aumentar el rendimiento del programa. El rendimiento se puede mejorar más todavía haciendo uso archivos de índices y luego cargar esa información a memoria, de esta manera se logra disminuir un poco más el acceso a disco y la estructura más eficiente para almacenar los índices en memoria es el árbol b.