Universidad Tecnológica Centroamericana

UNITEC

Proyecto Final

Manejador simple de archivos de registros

Estudiante:

Edilson Fernando González 11211070

Organización de Archivos, Sección: 411

Catedrático: Ing. Carlos Arias

Tegucigalpa, M.D.C., 23 de septiembre de 2013

Contenido

Introducción 4

Marco Teórico 5

Archivo 5

Archivo de registros 5

Avail List 5

Índices 6

QT 6

Implementación 7

ADTFile.h y ADTFile.cpp 8

ADTRecordFile.h y ADTRecordFile.cpp 10

Field.h y Field.cpp 13

MainWindow.h y MainWindow.cpp 14

Object.h y Object.cpp 18

PrimaryIndex.h y PrimaryIndex.cpp 18

Record.h y Record.cpp 19

Muestra de corrida y manual de usuario 20

Nuevo archivo 20

Abrir un archivo 21

Importar desde XML 22

Importar desde JSON 23

Crear campos 24

Modificar campos 24

Listar campos 25

Insertar registros 25
Buscar Registros 26
Eliminar registros 27
Listar registros 28
Exportar XML 29
Exportar JSON 30
Experiencia con Git 31
Conclusiones 31

Introducción

En el amplio mundo empresarial, hace algunas décadas, surgió producto del crecimiento asombroso del volumen de datos la necesidad de administrar archivos con registros que contuvieran información importante para almacenar de las diversas entidades u objetos que las empresas administran hoy en día.

Con esto, la ciencia de la computación tuvo que hacerse cargo en automatizar y optimizar el proceso de mantenimiento de tales registros, hasta llegar a concebir los poderosos DBMS que conocemos hoy en día.

El presente proyecto aplica técnicas básicas de manejo de registros en archivos de texto, se espera que el lector conozca de antemano el lenguaje C++ y su forma de manejar archivos de texto. Además se expondrán las técnicas utilizadas para minimizar la fragmentación de archivos y manejo de archivos con tamaños superiores a la memoria.

Puede obtener el código fuente del proyecto en la página web https://github.com/efgm1024/ProyectoOrga.

Marco Teórico

Para la alta comprensión del documento y el programa es necesario que el lector conozca a profundidad los conceptos que tienen que ver con archivos y su mantenimiento, además de las técnicas básicas y como logran implementarse. Un breve resumen es expuesto en los siguientes párrafos.

Archivo

Un archivo puede ser definido como un conjunto de bits que se guardan en un dispositivo secundario para que tenga carácter perdurable. Pueden clasificarse en dos tipos: lógicos y físicos. Los archivos lógicos son aquellos que se almacenan momentáneamente en memoria RAM y que conectan a un archivo físico en un dispositivo de almacenamiento secundario; cuando el sistema operativo considera conveniente realiza un vaciado de esta información al archivo físico, a menos, de que se realice una operación de vaciado forzando al sistema operativo a romper sus rutinas de optimización. Los archivos físicos son aquellos bits almacenados en los dispositivos perdurables como los discos duros, discos sólidos, cintas magnéticas, discos ópticos, entre otros.

Archivo de registros

Para definir un archivo de registros es necesario conocer de antemano el concepto de campo y de registro. Un campo es la unidad atómica de información que almacena datos en particular. Un conjunto de éstos forman un registro, así podemos definir un archivo de registros como un conjunto de bits que tienen una representación de los registros que están almacenados en él. Pueden existir archivos con registros de longitud fija, donde cada campo almacene información hasta un límite bien definido, o de longitud variable, donde no podemos predecir el tamaño de los datos de entrada.

Avail List

A partir de los problemas de fragmentación interna y desperdicio de espacio que existe en los archivos se creó una técnica para poder aprovechar al máximo éstos espacio que ya no tienen valor dentro del archivo. Así surge el concepto de Avail List, que posee una lista con los espacios disponibles dentro del archivo para poder utilizarlo en la inserción de nueva

información. Así puede evitarse con mayor precisión, aunque en ciertos casos no completamente, la fragmentación y la pérdida de espacio dentro del archivo.

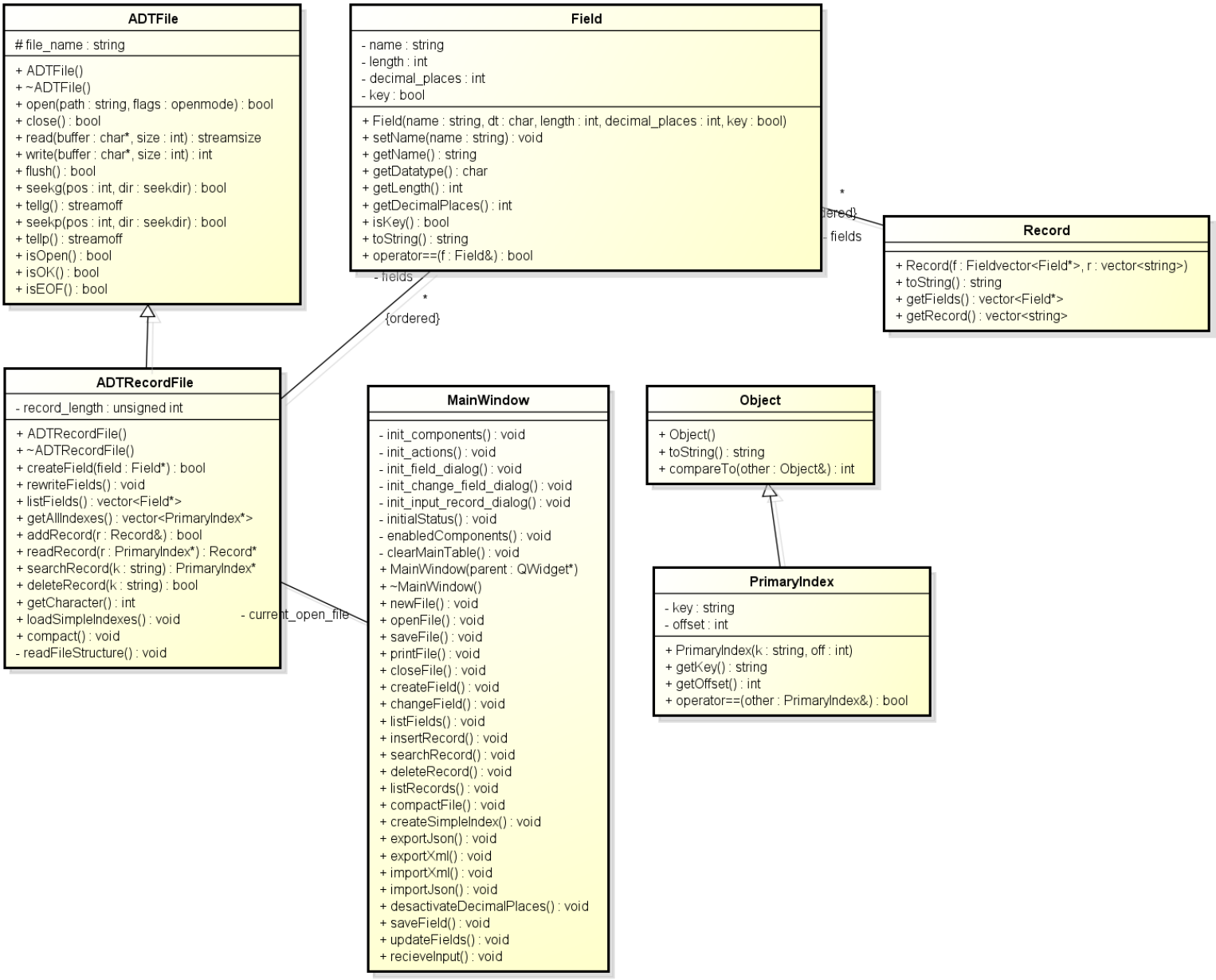
Índices

El índice, en su forma más general, es definido como una estructura de datos ordenada que mejora el rendimiento de las operaciones con archivos, valiéndose de llaves para identificar cada uno de los registros almacenados. Estos pueden clasificarse en primarios y secundarios. Los índices primarios, son aquellos que identifican un registro inequívocamente y son irrepetibles.

QT

QT es una librería de clases originalmente orientada a C++ pero ampliable a través de conexiones con otros lenguajes de programación como Python, Ruby, entre otros. Incluye elementos de interfaz gráfica, manejo de tipos compuestos, manejo de bases de datos, hilos, bibliotecas para el manejo de archivos XML y JSON.

Implementación



ADTFile.h y ADTFile.cpp

Clase padre que maneja las operaciones elementales con archivos.

Elemento	Descripción
`string file_name`	Nombre del archivo
`fstream fs`	Instancia de fstream que conecta el archivo físico con el archivo lógico
`ios_base::openmode flags`	Modos con los que fue abierto el archivo
`ADTFile()`	Constructor por defecto de la clase ADTFile
`~ADTFile()`	Destructor de la clase ADTFile, se encarga que al momento de destruir un ADTFile este haga un flush y se cierre
`open(string path, ios_base::openmode flags)`	Método que se encarga de abrir un archvo. Require como parámetros la dirección y el modo en que será abierto el archivo. Retorna true si el archivo es abierto con éxito, false en caso contrario.
`close()`	Método que se encarga de cerrar el archivo. Realiza un flush antes de cerarlo. Retorna true si fue cerrado con éxito, false caso contrario.
`read(char* buffer, int size)`	Método que lee información del archivo. Verifica si el archivo fue abierto para lectura. Debe de suministrarse como parámetros el buffer donde se almacenará la información leída y la cantidad de bytes que se leeran del archivo. Retorna la cantidad de bytes leidos que puede ser igual o menor que la cantidad de bytes

	enviados como parámetro. Retorna -1 si la lectura del archivo falló.
`write(const char* buffer, int size)`	Método que escribe información en un archivo. Verifica que el archivo fue abierto en modo de escritura. Se le suministra como parámetros el buffer donde ser desea escribir y la cantidad de bytes del buffer que se escribirán en el archivo. Si son escritos correctamente retorna la candidad de bytes enviada como parámetro, caso contrario retorna -1.
`flush()`	Método que vacía el contenido en memoria del archivo lógico hacia el archivo físico. Retorna un booleano si la operación se realizó con o sin éxito.
`seekg(int pos, ios_base::seekdir dir)`	Método que mueve el apuntador de lectura en un archivo. Se tiene que especificar la cantidad de bytes que se mueve y en que dirección. Las direcciones posibles son ios_base::beg (desde el inicio), ios_base::curr (desde la posición actual), ios_base::end (desde el final del archivo). Retorna un booleano si la operación se realizó con o sin éxito.
`tellg()`	Método que retorna la posición en la que está el apuntador de lectura detro del archivo.
`seekp(int pos, ios_base::seekdir dir)`	Método que mueve el apuntador de escritura en un archivo. Se tiene que

	especificar la cantidad de bytes que se mueve y en que dirección. Las direcciones posibles son ios_base::beg (desde el inicio), ios_base::curr (desde la posición actual), ios_base::end (desde el final del archivo). Retorna un booleano si la operación se realizó con o sin éxito.
`tellp()`	Método que retorna la posición en la que está el apuntador de escritura detro del archivo.
`isOpen()`	Método que retorna un booleano si el archivo está o no abierto.
`isOK()`	Método que retorna un booleano si el archivo está bien o las banderas de error están activadas.
`isEOF()`	Método que retorna un booleano de si el archivo está o no al final de él.

ADTRecordFile.h y ADTRecordFile.cpp

Maneja las operaciones elementales con archivos de registros: CRU de campos, CRUD de registros y el manejo de índices por medio de mapas.

Elemento	**Descripción**
`const char HEADER_END = '&'`	Constante para identificar el final del file header
`const char DELETED = '?'`	Constante para identificar a un registro eliminado

`vector<Field*> fields`	Vector de campos que almacena los campos que contiene el archivo de registros
`unsigned int record_length`	Tamaño de un registro individual
`streamoff begin_body`	Offset donde comienza el cuerpo del archivo de registros
`QMap<QString, PrimaryIndex*> indexes`	Mapa con los índices simples
`QStack<streamoff> avail_list`	Avail List para controlar los espacios disponibles dentro de un archivo.
`ADTRecordFile()`	Constructor por defecto de ADTRecordFile, lee la estructura del archivo para inicializar los campos además de cargar los índices para inicializar los índices simples.
`~ADTRecordFile()`	Destructor de ADTRecordFile
`readFileStructure()`	Método que se encarga de leer la estructura del header del archivo. Inicializa el vector de campos con su información pertinente.
`loadSimpleIndexes()`	Método que se encarga de levantar a memoria los índices simples
`listFields()`	Método que actualiza la estructura del archivo y retorna los campos de éste.
`createField(Field* field)`	Método que añade campos al archivo de registros
`rewriteFields()`	Método que actualiza los campos en el archivo de registros
`getCharacter()`	Método que obtiene un caracter del archivo, si falla su lectura retorna -1

`addRecord(Record& r)`	Método que añade un registro al archivo de registros.
`getAllIndexes()`	Método que actualiza los índices del archivo y retorna un vector de éstos
`readRecord(PrimaryIndex* r)`	Método que lee un registro del archivo. Como parámetro recibe el índice primario identifica al registro
`searchRecord(string k)`	Método que busca un registro utilizando índices simples en un mapa. Retorna NULL si la clave del registro no fue encontrada en el archivo de registros. La clave es suministrada como parámetro de tipo string. Retorna el un PrimaryIndex con la clave y el offset donde se encuentra el registro ubicado en el archivo.
`deleteRecord(string k)`	Método que marca un registro como eliminado dentro del archivo de registros. Recibe como parámetro la clave del en formato string. Agrega el espacio disponible al avail List si la operación se completó con éxito, en este caso retorna true, caso contrario retorna false.
`compact()`	Método que se encarga de eliminar definitivamente los registros marcados como eliminados con anterioridad en las operaciones de archivos con registros.

Field.h y Field.cpp

Almacena la información necesaria para un campo de un archivo de registros.

Elemento	Descripción

`typedef char datatype;`
`const datatype INT_DT = 'i';`
`const datatype REAL_DT = 'r';`
`const datatype STRING_DT = 'c';`
	Tipos de datos

`const int FIELD_LENGTH = 30;`
`const int DATA_TYPE_LENGTH = 1;`
`const int LENGTH_LEGTH = 3;`
`const int DECIMAL_PLACES_LEGTH = 3;`
`const int KEY_LEGTH = 1;`
	Tamaños de los atributos

`const string INT_DVALUE = "0";`
`const string REAL_DVALUE = "0.0";`
`const string STRING_DVALUE = "NULL";`
	Valores por defecto de los tipos

`string name`	Nombre del campo
`datatype data_type`	Tipo de datos del campo
`int length`	Tamaño del campo
`int decimal_places`	Cantidad de espacios decimales
`bool key`	Es llave o no
`Field(string name, datatype dt, int length, int decimal_places, bool key)`	Constructor de la clase campo, crea un campo con la información necesaria para guardar en un archivo de texto. Recibe como parámetros el nombre, el tipo de dato (INT_DT, STRING_DT y REAL_DT) la longitud, la cantidad de espacios

	decimales y un booleano de si es o no llave.
`setName(string name)`	Método mutador de la clase campo que modifica el nombre del campo, Solo se almacenan los primeros 30 caracteres del nombre por cuestiones de espacio.
`getName()`	Método accesor para el nombre del campo.
`getDatatype()`	Método accesor para el tipo de datos del campo. Los valores retornados pueden ser INT_DT, STRING_DT y REAL_DT.
`getLength()`	Método accesor del tamaño del campo.
`getDecimalPlaces()`	Método accesor de la cantidad de espacios decimales del campo.
`isKey()`	Método predicado que retorna un boleano para identificar un campo llave.
`toString()`	Método que retorna un string con la representación en cadena para ser escrita en un archivo de registros
`operator==(const Field& f)`	Sobrecarga de operador para comparar la igualdad entre dos campos.

MainWindow.h y MainWindow.cpp

Ventana principal del programa

Elemento	**Descripción**
`const QString PROGRAM_NAME = "Edilson Fernando Gonzalez";` `const QString EXTENSION = ".edb";`	Contantes de metadatos

`init_components()`	Método que incializa todos los componentes gŕaficos de la ventana
`init_actions()`	Método que se encarga de inicializar las acciones u opciones de todos los menús de la interfaz gráfica.
`init_field_dialog()`	Método que se encarga de inicializar los componentes gráficos de la ventana de ingreso de campos.
`init_change_field_dialog()`	Método que se encarga de inicializar los componentes de la ventana de actualizar campos
`init_input_record_dialog()`	Método que se encarga de inicializar los componentes gráficos de la ventana de ingreso de registros
`initialStatus()`	Método que se encarga de desactivar las opciones de menú innecesarias para comenzar a utilizar el programa
`enabledComponents()`	Método que reactiva las opciones de menú para comenzar a utilizar el programa.
`clearMainTable()`	Método que se encarga de eliminar la información que tenía la tabla principal anteriormente en el programa
`newFile()`	Método que se encarga de crear un nuevo archivo y abrirlo para su uso en el programa
`openFile()`	Método encargado de abrir archivos de registros ya existentes
`saveFile()`	Método que se encarga de guardar el archivo a través de una llamada a flush.

printFile()	Método que imprime el archivo actual de registros en un archivo PDF
closeFile()	Método que cierra un archivo de registros abierto y desactiva los componentes de los menús.
createField()	Método encargado de levantar la ventana que crea campos. Si ya exiten registros en el archivo no es posible crear más campos.
changeField()	Método que lanza una ventana que permite actualizar solamente el nombre de los campos ya ingresados dentro del archivo del registros
listFields()	Método que se encarga de listar los campos que posee el archivo de registros
insertRecord()	Método que se encarga de recolectar la información de un registro y añadirla al archivo de registros
searchRecord()	Método que busca en un registro y lo despliega en la tabla principal si es encontrado.
deleteRecord()	Método encargado de eliminar un registro a partir de la clave suministrada por el usuario.
listRecords()	Método que lista los registros en la tabla principal del programa
compactFile()	Método que llama a compactar archivo de registros
createSimpleIndex()	Método que llama a la creación de índices simples en un archivo de registros

exportJson()	Método que se encarga de exportar el actual archivo de registros en un archivo JSON
exportXml()	Método que exporta el actual archivo de regisros al formato XML
importXml()	Método que lee un archivo XML y lo transfiere a un nuevo archivo de registros
importJson()	Método que importa un archivo JSON a los archivos de registro del programa
desactivateDecimalPlaces()	Método que se encarga de desactivar el ingreso de espacios decimales en los casos donde no se requiera el uso.
saveField()	Método que envía la información obtenida de un campo a través de las ventanas al archivo de registros
updateFields()	Método que actualiza el nombre de los campos a partir de la información obtenida de la respectiva ventana ingresada por el usuario.
recieveInput()	Método que valida que los datos recibidos por el usuario sean consistentes y no vacios.

Object.h y Object.cpp

Clase general de donde heredan los objetos que funcionarían en los índices y el árbol B

Elemento	Descripción
`Object()`	Constructor por defecto de la clase object
`toString()`	Método que retorna una representación string sencilla de un object
`compareTo(const Object&)`	Método que compara dos objetos. Retorna 1 si this es mayor que el objeto parámetro, -1 si this es menor que el objeto parámetro y 0 si ambos son iguales.

PrimaryIndex.h y PrimaryIndex.cpp

Clase que maneja información para un índice primario. Se utilizará para el manejo de índices simples e índices de árbol B.

Elemento	Descripción
`string key`	Llave correspondiente al índice primario
`int offset`	Offset en donde se encuentra el registro
`Object()`	Constructor por defecto de la clase object
`PrimaryIndex(string k, int off)`	Contructor de la clase que recibe un string con la clave y un int con el offset donde se encuentra el registro que identifica el índice primario
`getKey()`	Método accesor para la llave del índice.
`getOffset()`	Métoodo accesor para el offset almacenado en el índice primario
`operator ==(const PrimaryIndex&)`	Sobrecarga de operador que retorna si dos índices primarios son iguales

Record.h y Record.cpp

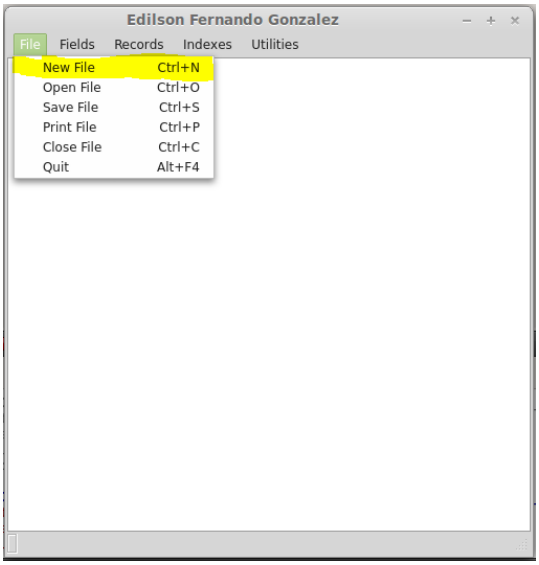
Almacena información de los campos y el registro que será almacenado en archivos. Posee la estructura necesaria para ser transferido a texto a partir de la información suministrada en un vector de campos con sus características.

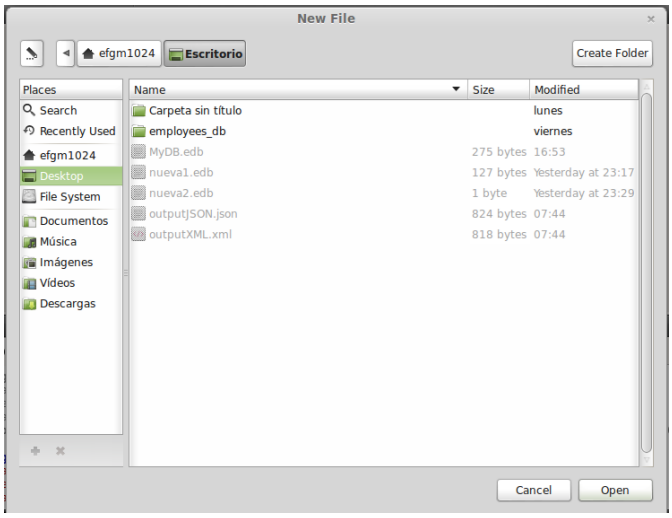
Elemento	**Descripción**
`vector<Field*> fields`	Vector que contiene la información de los campos del registro
`vector<string> record`	Información propia del registro.
`Record(vector<Field*> f, vector<string> r)`	Constructor por defecto que recibe un vector de campos otro vector de strings con los valores del registro.
`toString()`	Método que retorna una representación string del objecto registro para ser escrito en un archivo
`getFields()`	Método accesor para el vector de campos.
`getRecord()`	Método accesor para el vector de registro en string.

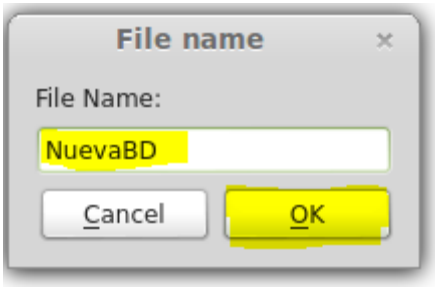
Muestra de corrida y manual de usuario

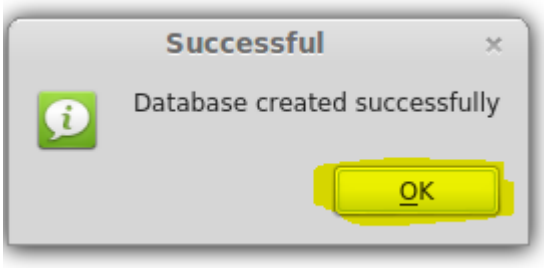
Nuevo archivo

File | New File | Seleccione la carpeta donde desea guardar el nuevo archivo | Open | Ingrese el nombre de la base de datos | OK | Mensaje de éxito | OK



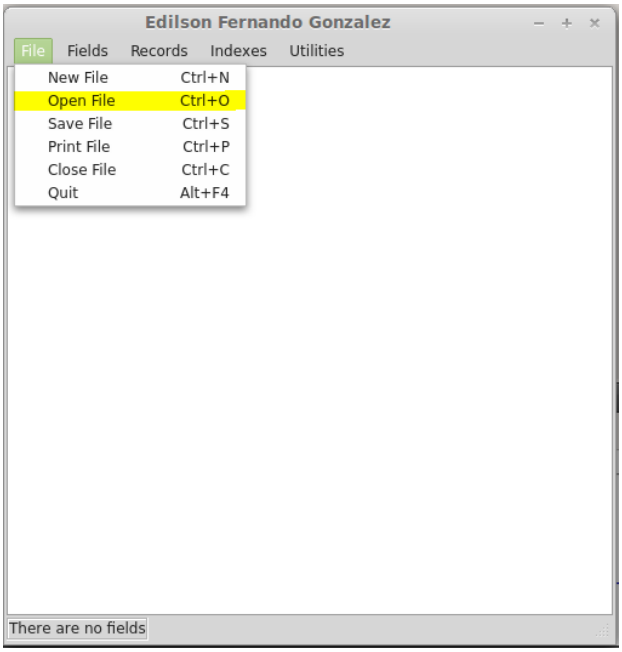


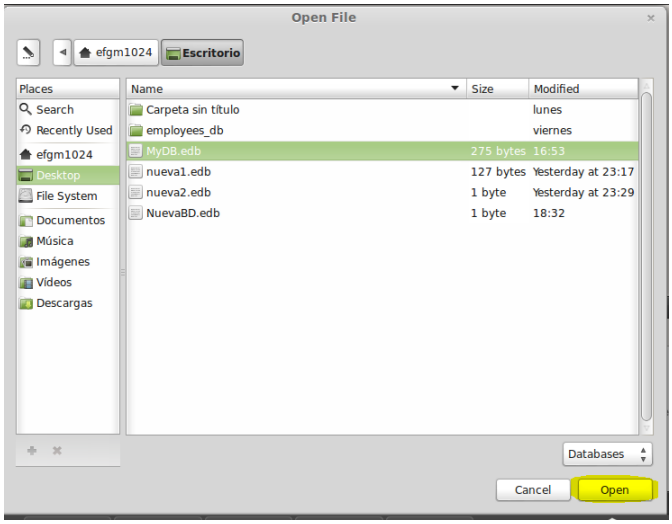


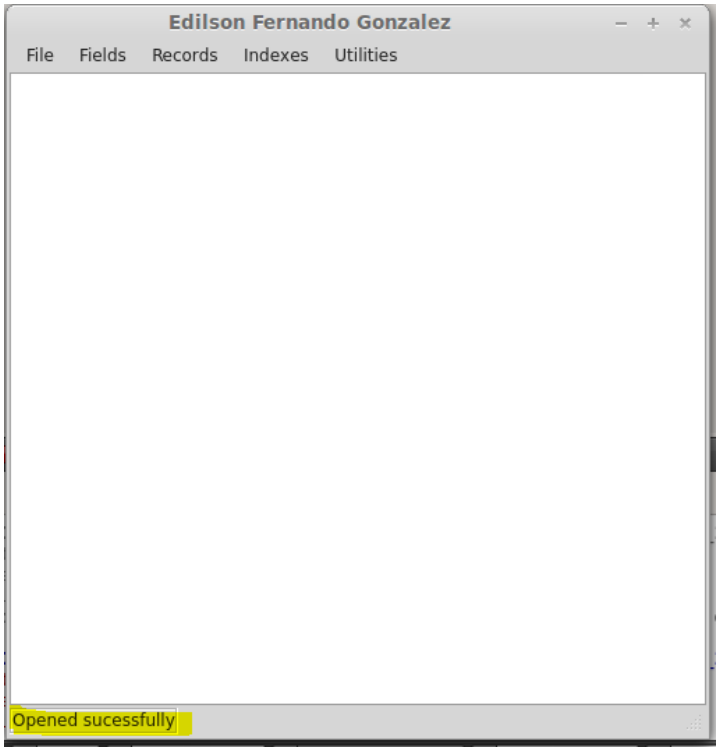


Abrir un archivo

File | Open File | Seleccione la base de datos (extensión .edb) | Open | Verifique el mensaje en la status bar

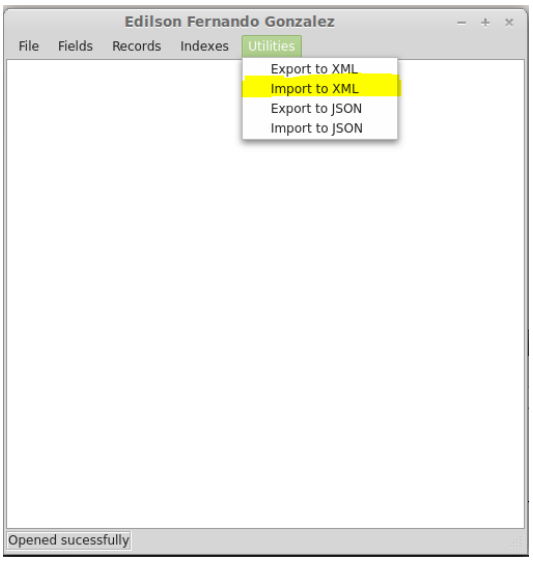


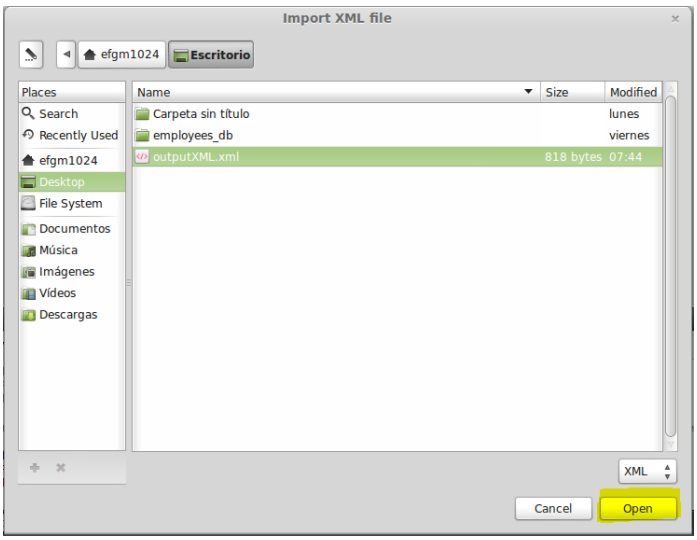


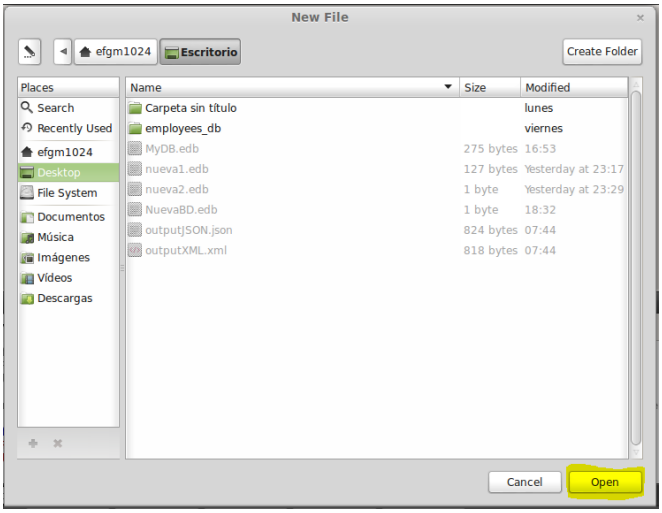


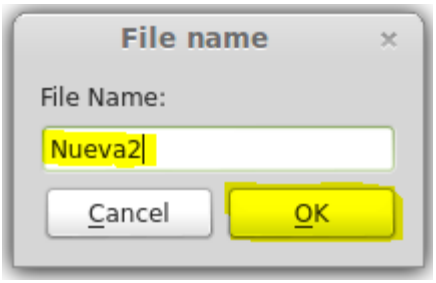
Importar desde XML

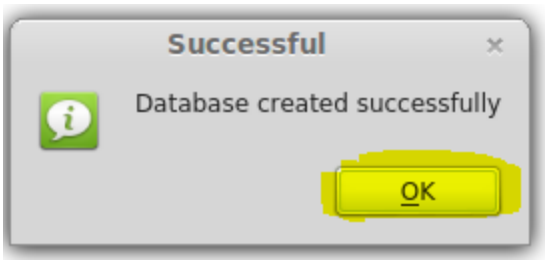
Utilities | Import from XML | Seleccione el archivo XML | Open | Seleccione la ruta del nuevo archivo (extensión .edb) | Coloque el nombre del nuevo archivo | OK | Mensaje de confirmación | OK





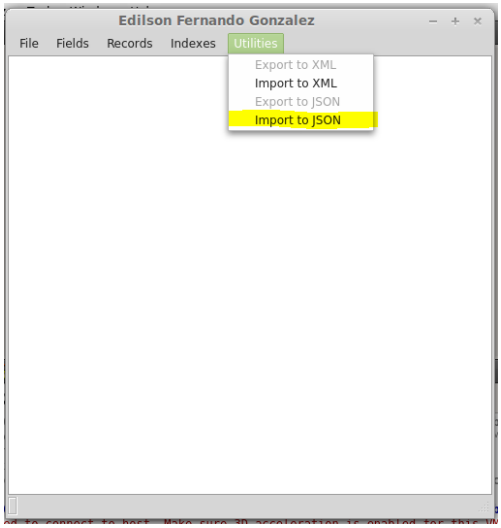


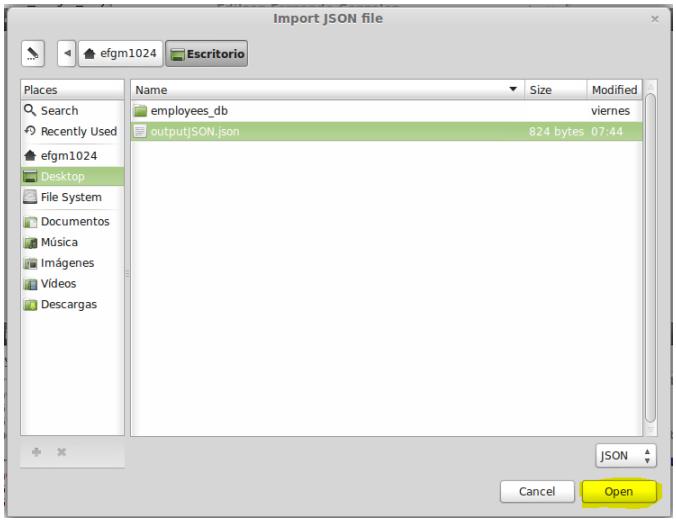


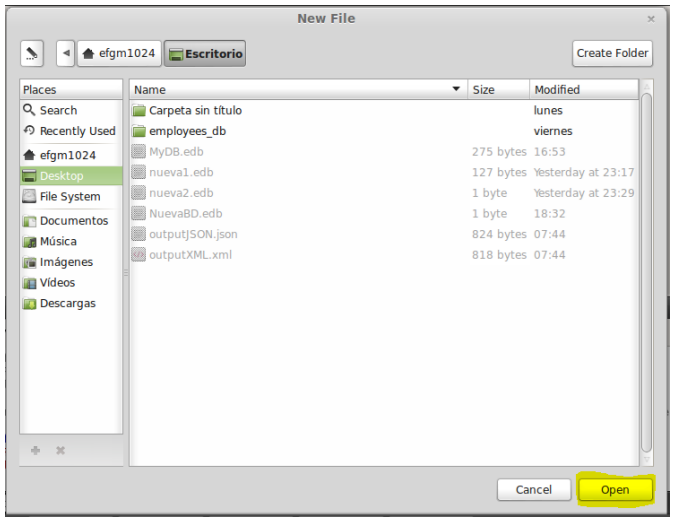


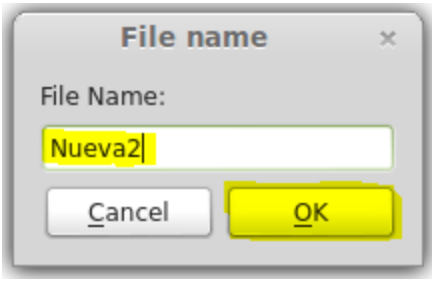
Importar desde JSON

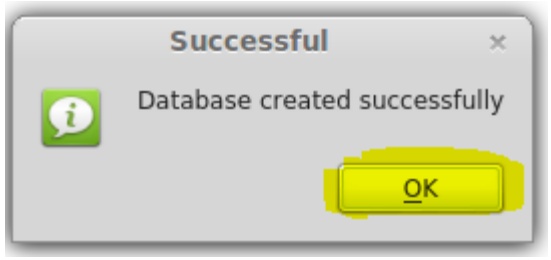
Utilities | Import from JSON | Seleccione el archivo JSON que se desea importar | Open | Seleccione la carpeta donde se va a guardar el nuevo archivo (extensión .edb) | Coloque el nombre del nuevo archivo | OK | Espere el mensaje de confirmación | OK





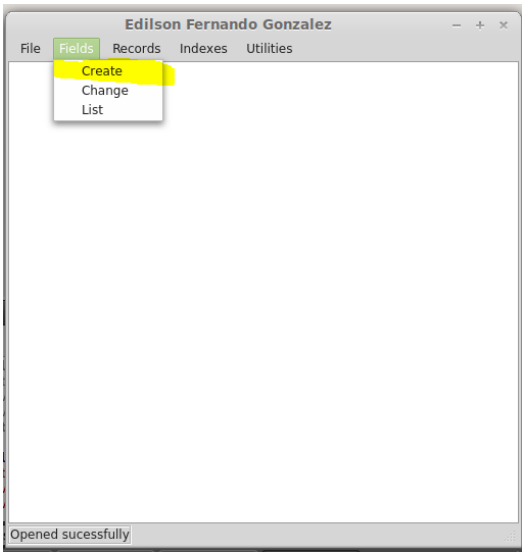


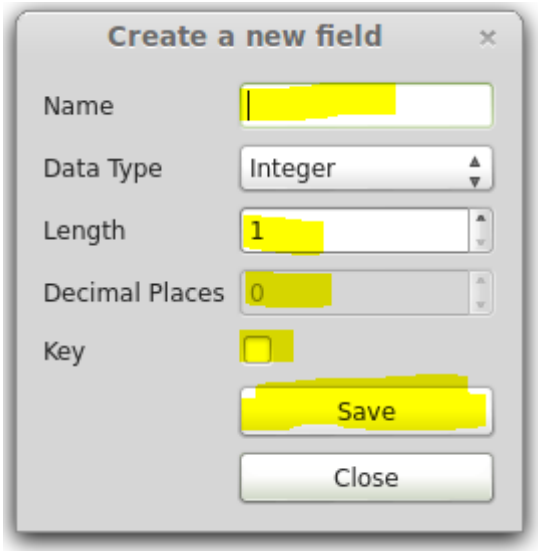




Crear campos

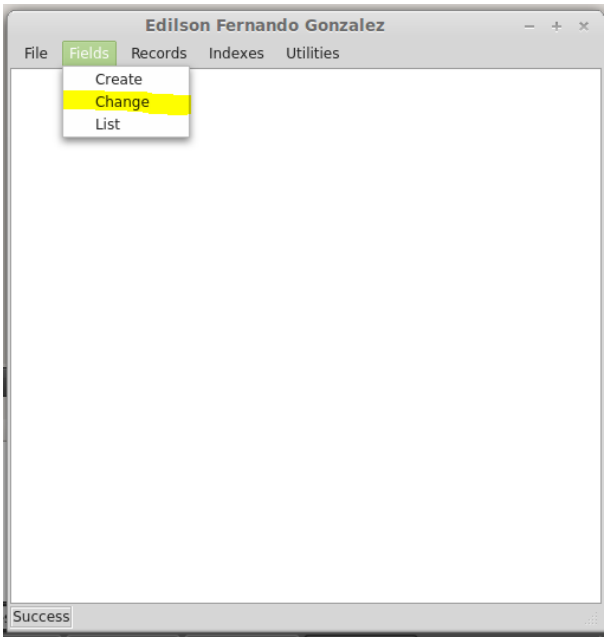
Fields | Create | Rellene la información solicitada por la ventana | Save | Realice el proceso cuantas veces sea necesario | Close |

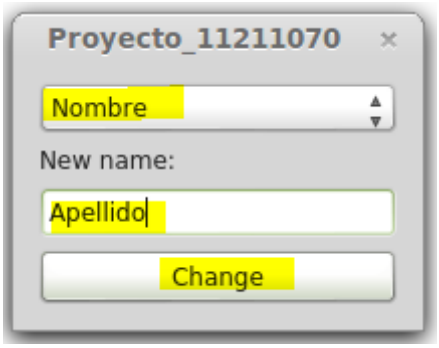




Modificar campos

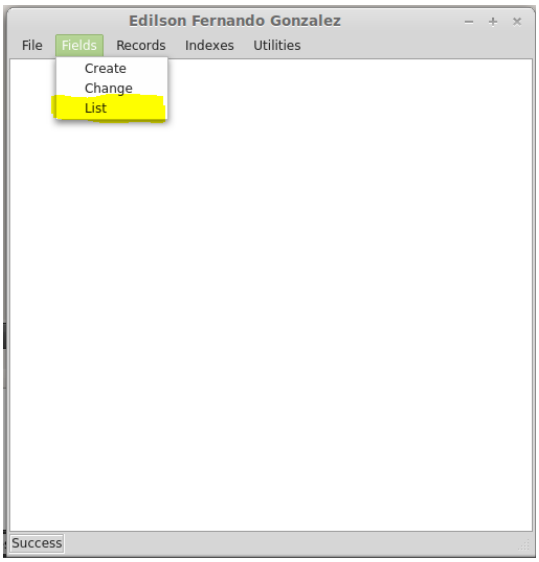
Fields | Change | Seleccione el campo que desea modificar | Escriba el nuevo nombre del campo | Change

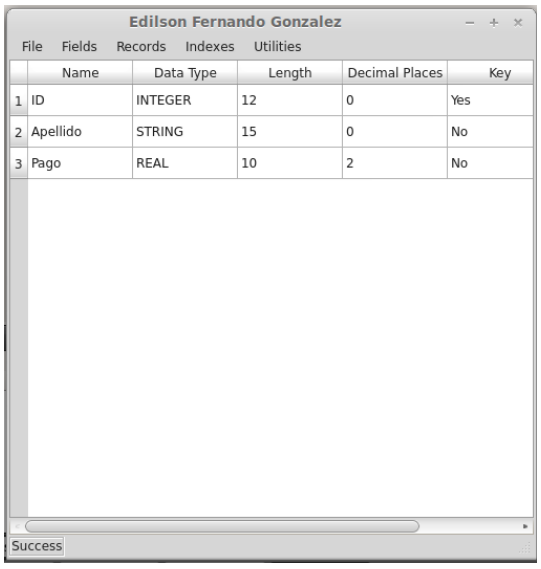




Listar campos

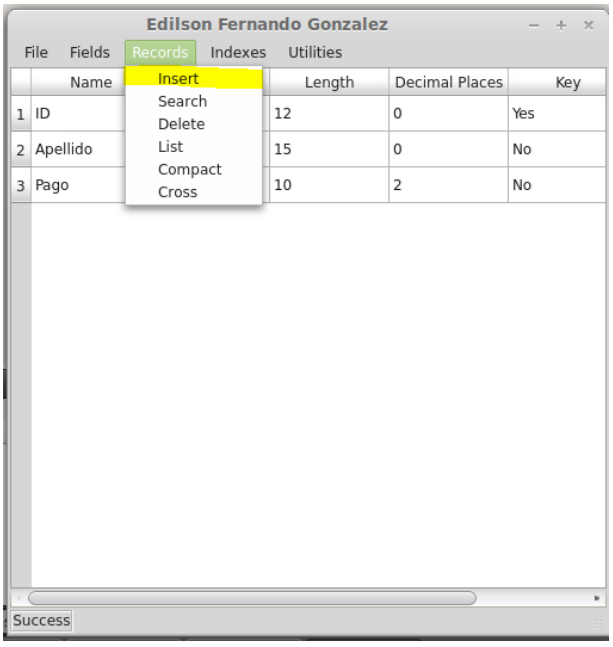
Fields | List | Compruebe la pantalla principal para observar la información de los campos del archivo

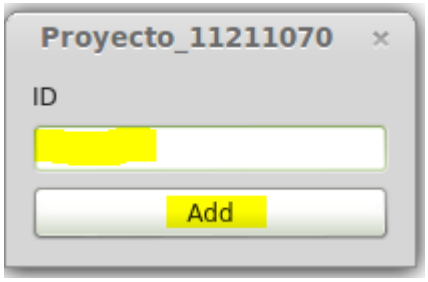


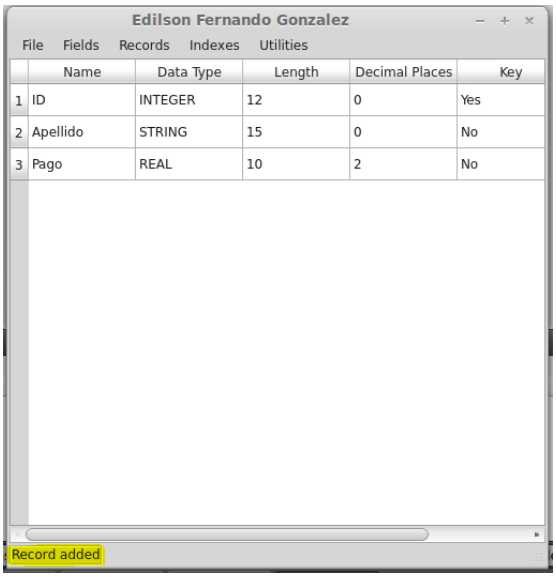


Insertar registros

Records | Insert | Rellene el campo con la información solicitada | Add | repita el proceso para cada uno de los campos | Observe la Status Bar para obtener información

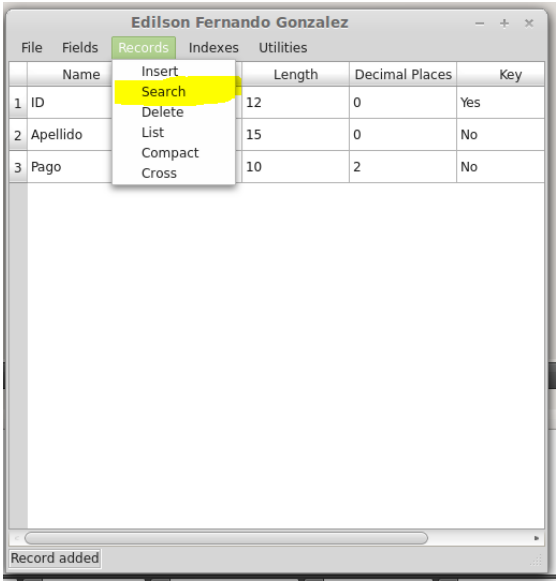


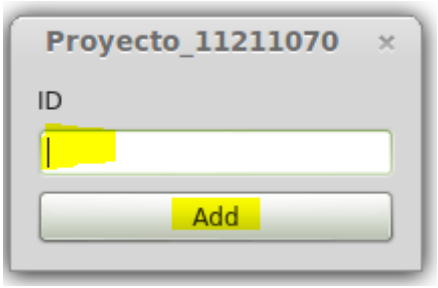


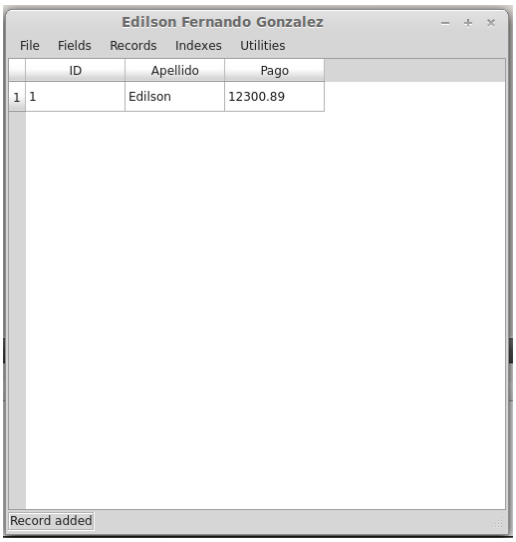


Buscar Registros

Records | Search | Ingrese la información solicitada del registro que desea buscar | Add | Repita el proceso para los campos solicitados | Observe la pantalla principal con el resultado

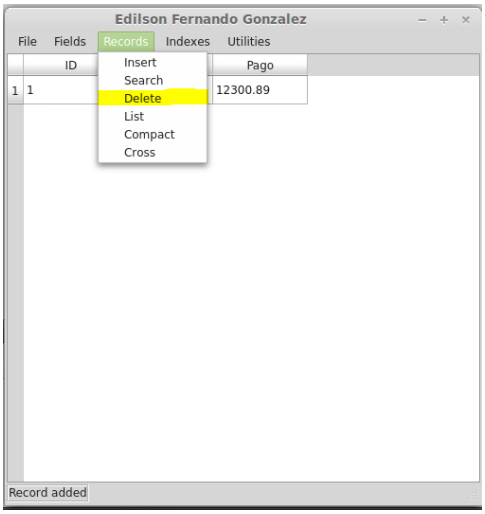


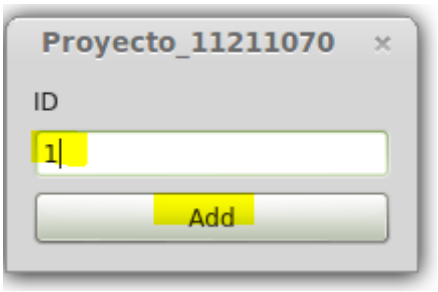


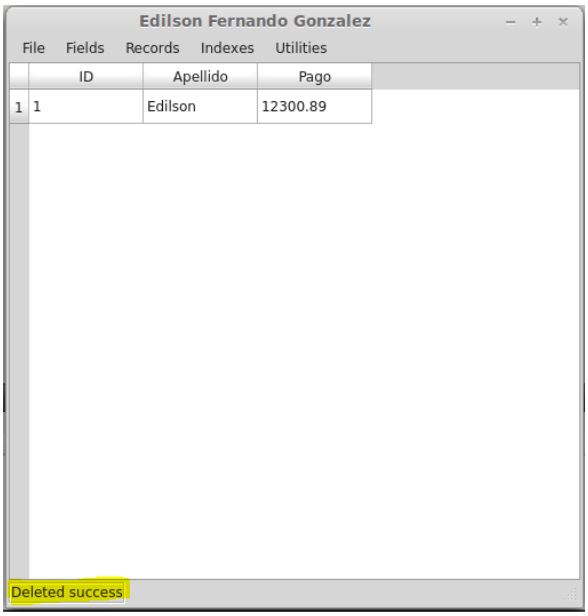


Eliminar registros

Records | Delete | Ingrese la información solicitada del registro que desea eliminar | Add | Repita el proceso para los campos solicitados | Observe la status bar con el resultado

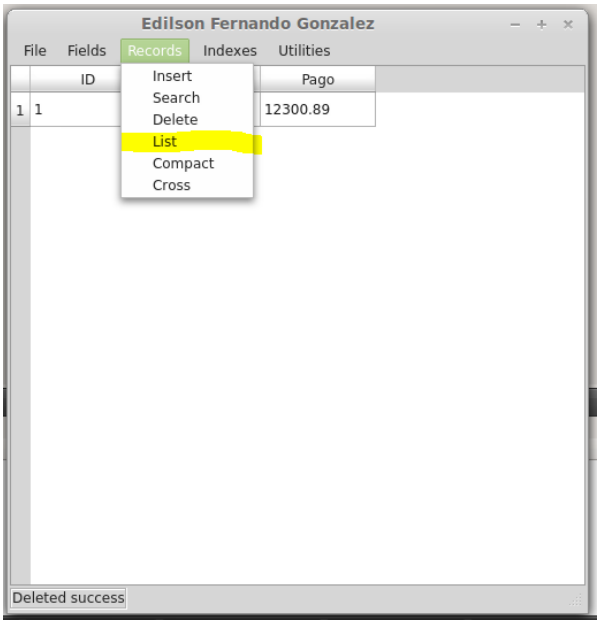


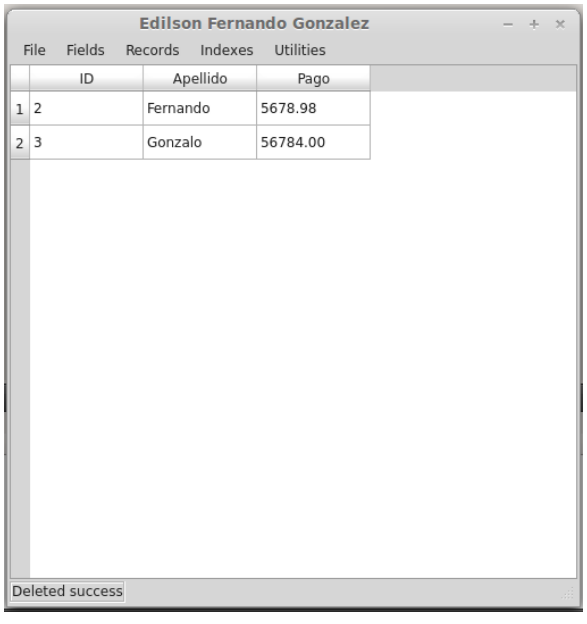




Listar registros

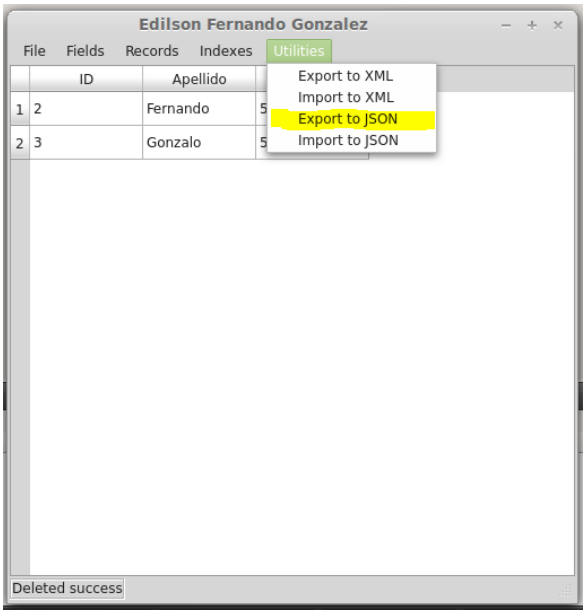
Records | List | Observe el resultado en la pantalla principal

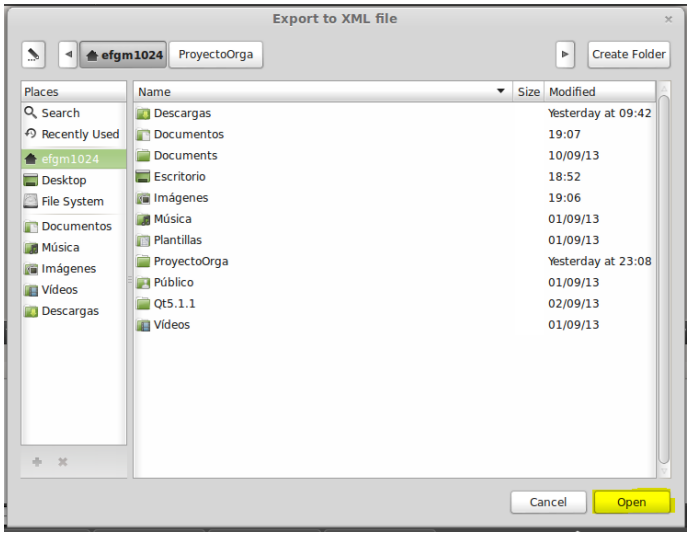


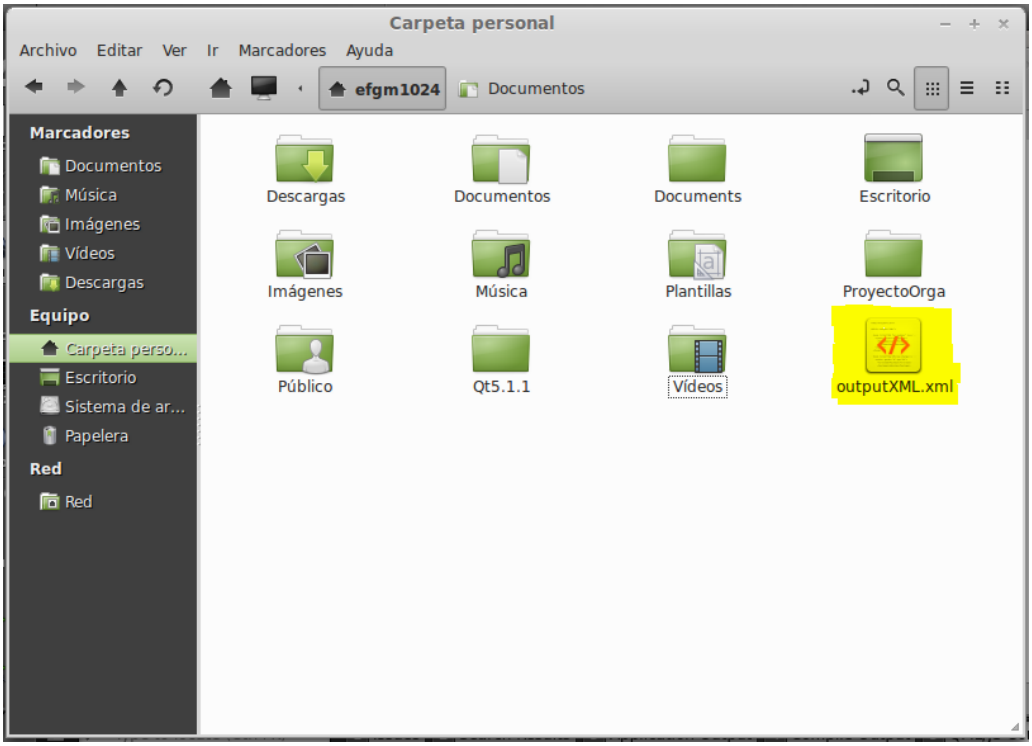


Exportar XML

Utilities | Export to XML | Seleccione la carpeta donde desea guardar el archivo XML | verifique la carpeta para observar el resultado

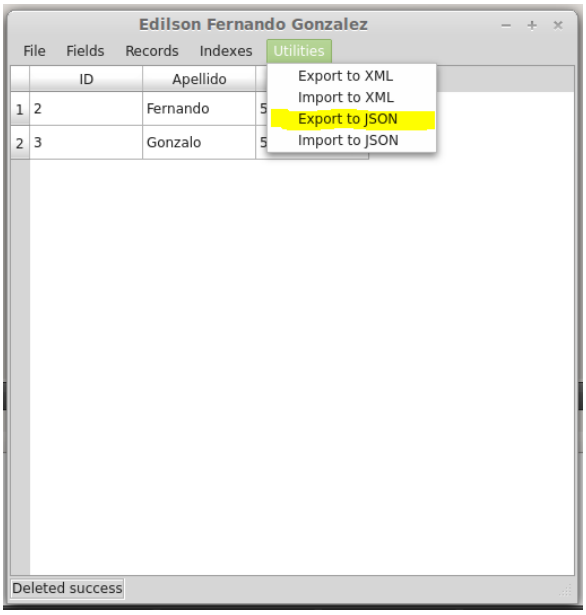


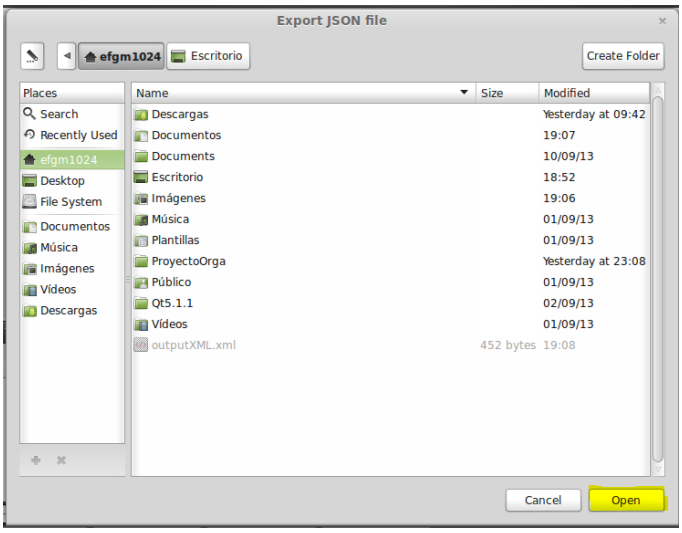


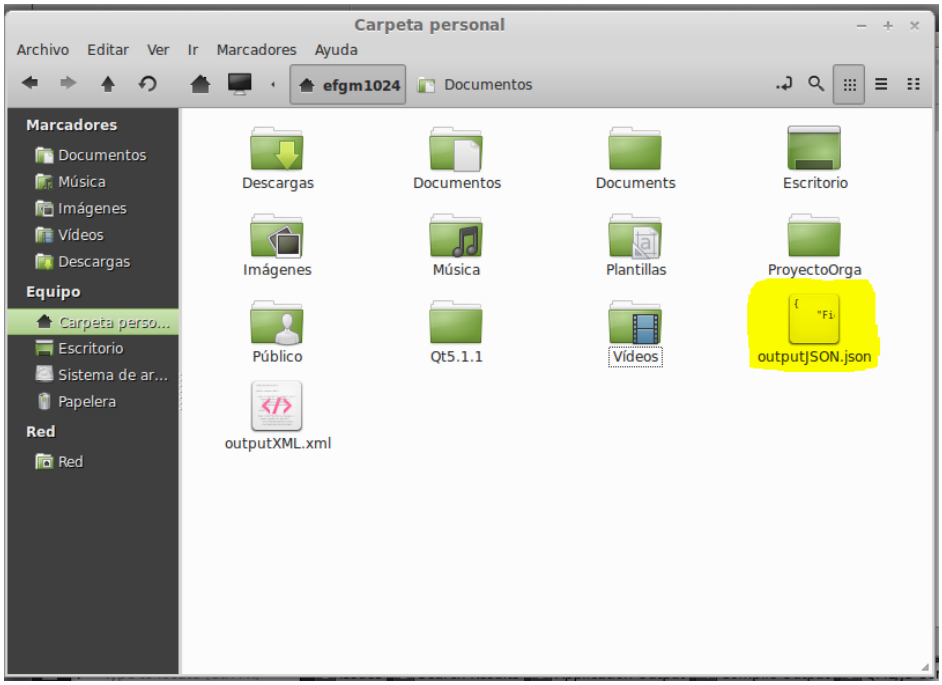


Exportar JSON

Utilities | Export to JSON | Seleccione la carpeta donde desea guardar el archivo JSON | verifique la carpeta para observar el resultado







Experiencia con Git

Mi experiencia de trabajo con la herramienta ha tenido sus impases en cuanto a trabajo en grupo. El programa suele agregar líneas de error cuando dos archivos modificados por otras personas no coinciden con el propio, causando así pérdidas de información y descontrol generado por las líneas de más.

Además, los repositorios pueden ser restaurados y reiniciados con facilidad sin muchas advertencias. Posiblemente los problemas hayan radicado en el desconocimiento del uso de la herramienta, aunque hay muchos manuales, considero que la interfaz por consola es poco intuitiva y no muy explicativa. El cliente con interfaz gráfica más amigable se reserva para sistemas como Windows y Mac OS, para Linux no existen alternativas.

Por estar razones, en conclusión, es completamente aceptable afirmar que la experiencia dejó de ser grata y duradera para el trabajo fructífero en equipo.

Conclusiones

El trabajo con archivos tiende a ser engorroso y repetitivo, como buena práctica de programación y una técnica para agilizar el desarrollo de software, se recomienda encarecidamente utilizar abstracciones de archivos más potentes como las de QT o las de Java, debido a que las librerías de C++ poseen demasiados errores y problemas entre plataformas diferentes.

Las técnicas para manejo de archivos de registros de enormes proporciones conllevan muchos puntos a tratar, aunque la correcta optimización de archivos pequeños favorece el rendimiento en dispositivos donde es crucial evitar un alza de tiempo de procesamiento como ser los dispositivos móviles. Así es posible diseñar pequeñas bases de datos portables evitando el consumo de batería y evitando la saturación de la memoria principal, secundaria y procesador.