Unitec

[DOCUMENTACIÓN PROYECTO ACCESS]

Ing. Carlos Arias

Daniel Alejandro Coello Martínez (11141303)

Ricardo Ortega (10941162)

Fecha de entrega: 23 de Septiembre de 2013

Tegucigalpa, M.D.C.

Contenido

Introducción

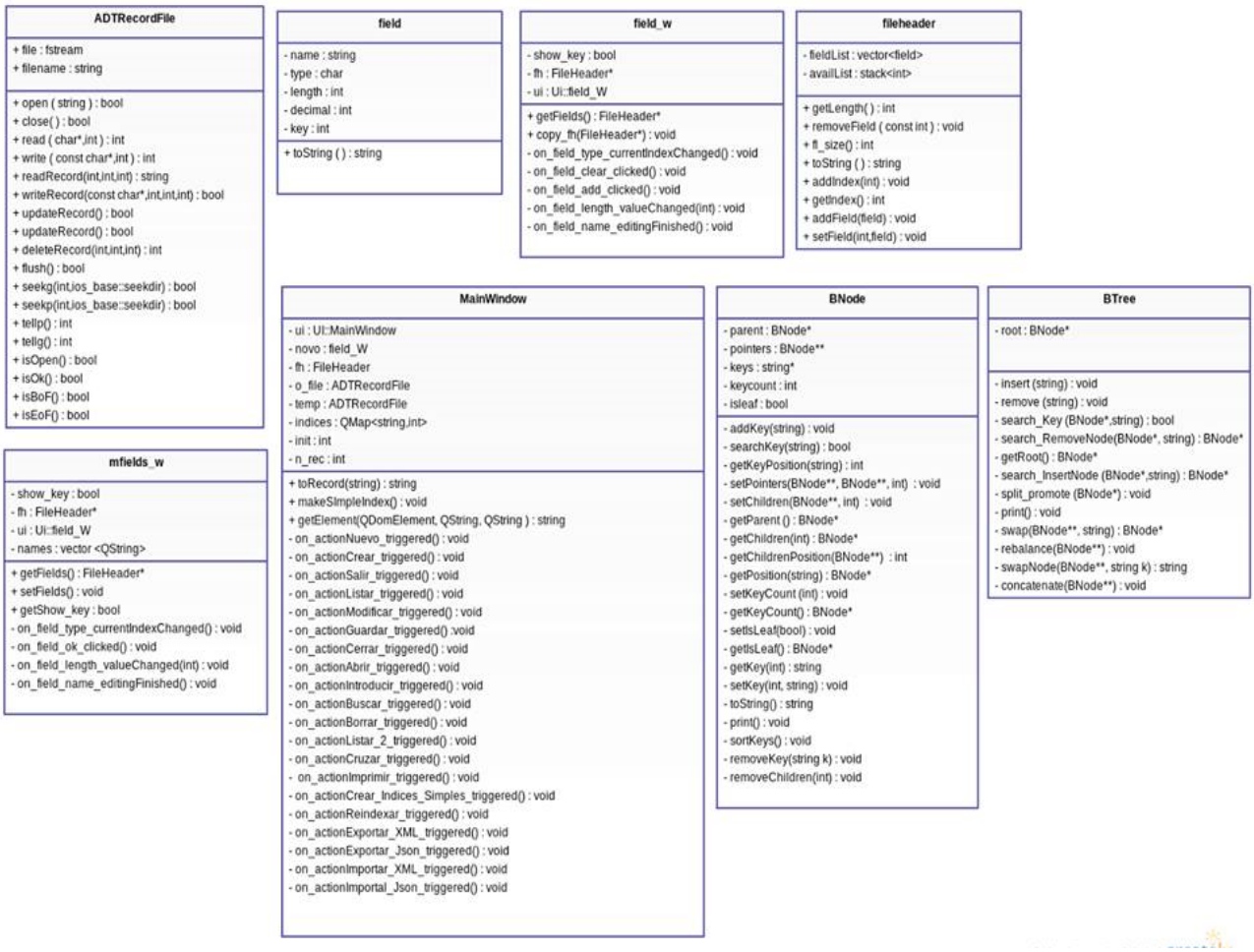
El proyecto es una simulación básica de la base de datos de Access. En esta base de datos, el usuario puede crear archivos de registros en los cuales podrá realizar las siguientes acciones:

- Crear campos
- Guardar campos
- Ingresar registros
- Guardar registros
- Crear índices
- Importar/Exportar XML
- Importar/Exportar JSON

El usuario primero tendrá que definir los campos que quiere en su archivo de registro, después podrá ingresar los datos a su archivo de registro y realizar alguna de las operaciones adicionales, ya sea de búsqueda o de eliminación.

Para este proyecto, se utilizó el lenguaje de programación de C++, el IDE QT Creator y la máquina virtual VM VirtualBox con Linux Mint 15 .

Marco Teórico



ADT RecordFile

En esta clase es en sí, el archivo de registros. Aquí es donde se hace la inserción, lectura, eliminación y actualización de registros. En esta clase es donde se llevan a cabo las operaciones IO del programa al momento de manipularse los registros.

BNODE

Esta es la unidad básica del árbol B. Aquí es donde se guardan todos los elementos de la página, utilizando dos arreglos como base. En el primer arreglo se guardan todas las llaves de la página. En el segundo arreglo se guardan todos los apuntadores de dichas llaves, que determinan los demás nodos que son mayores o menores que el nodo actual. Aquí se lleva a cabo la inserción de llaves, actualización de subárboles y eliminación de llaves.

BTree

Este es el árbol en sí. Se guarda un solo nodo raíz que tendrá las direcciones a todos los demás nodos. En esta clase, se llevará a cabo la inserción y el Split-promote, la eliminación y sus métodos de redistribución y concatenación.

Field

Esta clase modela los campos dentro del archivo de registro. Aquí se determina el tipo de campo, la longitud, si es llave, cuantas posiciones decimales tiene si es número y el nombre del campo.

Field_w

Esta clase se usa para modelar gráficamente la ventana de inserción de campos en QT. Aquí es donde se creó la interfaz gráfica para que el usuario cree los campos.

FileHeader

Esta clase es donde se modela el encabezado del archivo de registro. Aquí se guardan y eliminan los campos existentes y se actualiza el estado del AvailList para su exportación en el ADT RecordFile.

MainWindow

Aquí es donde se modela toda la ventana principal. Contiene toda la interfaz gráfica del proyecto que incluye todas las demás ventanas y clases. Es en esta ventana donde se lleva a cabo lo más importante y se ejecuta el programa en su totalidad.

mFields_w

En esta ventana, se pueden modificar los fields ya creados previamente en el proyecto.

Implementación

A continuación se explicará con más detalle los atributos y métodos de cada clase (constructores, mutadores y accesores no fueron agregadas por razones lógicas).

ADT RecordFile

Atributos

fstream file: es el archivo en sí

string filename: guarda el nombre del archivo

Funciones

bool open(string): Abre el archivo de registro

bool close(): Cierra el archivo de registro

int read(char*,int): lee del archivo y asigna lo leído a un string

int write(const char*,int): escribe en el archivo el string enviado

string readRecord(int,int,int): lee un registro

bool writeRecord(const char*,int,int,int): escribe un registro

bool updateRecord(): actualiza el archivo

int deleteRecord(int,int,int): elimina un registro

bool flush(): flushea el archivo

bool seekg(int,ios_base::seekdir): mueve el puntero de lectura

bool seekp(int,ios_base::seekdir): mueve el puntero de escritura

int tellp(): indica la posición del puntero de escritura

int tellg(): indica la posición del puntero de lectura

bool isOpen(): Determina si el archivo está abierto

bool isOk(): Determina si el archivo está en condiciones para operar con el

bool isBoF(): Indica si el puntero de lectura o escritura esta al principio

bool isEoF(): Indica si el puntero de lectura o escritura esta al final

Field

Atributos

string name: nombre del campo

char type: tipo del campo(INT=0,STRING=1,DOUBLE=2)

int length: LONGITUD del campo

int decimal: cuantos puntos decimales tendrá el campo

int key: Campo llave

Funciones

String toString(): retorna el toString de la clase

Field_w

Atributos

Ui::field_W *ui: Ventana de crear campos

FileHeader *fh: FH a devolver al cerrar la ventana

bool show_key: Bandera que muestra o no la llave

Funciones

void on_field_type_currentIndexChanged(int index): Hace las validaciones dependiendo de su tipo

void on_field_clear_clicked(): Limpia los atributos dados hasta el momento

void on_field_add_clicked(): Agrega un Campo al FH

void on_field_length_valueChanged(int arg1): Aplica lo mínimo o max de la longitud si es decimal

void on_field_name_editingFinished(): Verifica si está o no vacío el campo del nombre

FileHeader* getFields(): Devuelve los campos crados

void copy_fh(FileHeader*): copia un FH al de la clase

FileHeader

Atributos

stack<int> availList: Contiene los registros eliminados

vector<field> fieldlist: contiene los campos del archivo

Funciones

stack<int> getAL(): devuelve el AL

void addField (field): Agrega un campo

int fl_size()const: devuelve el tamanio del FL

vector<field> getFL()const: devuelve el FL

void removeField (const int): elimina un campo

const int getLength () const: obtiene la longitud por registro

void addIndex (const int): agrega un rrn al AL

int getIndex(): Obtiene el ultimo rrn del AL

string toString(): Devuelve el FH del archivo

mFields_w

Atributos

Ui::mFields_w *ui: Ventana de modificar

bool show_key: Bandera que muestra o no la llave

field f: Campo a modificar

vector<QString> names: Contiene los nombres de los campos

Funciones

void setField(field,vector<QString>): Modifica un campo

field getField(): Devuelve el campo modificado

bool getShow_key(): Muestra si es o no una llave

void on_mod_tipo_currentIndexChanged(int index): Hace las validaciones dependiendo de su tipo

void on_mod_length_valueChanged(int arg1): Aplica lo mínimo o max de la longitud si es decimal

void on_mod_nombre_editingFinished(): Verifica si está o no vacío el campo del nombre

void on_mod_ok_clicked(): Modifica el campo

Main Window

Atributos

Ui::MainWindow *ui: Ventana Principal

field_W *novo: Ventana de crear campos

FileHeader *fh: File Header del Archivo

ADTRecordFile* o_file: Archivo de Registros

ADTRecordFile* temp: Archivo temporal para compactar

QMap<string,int> indices: Índices Simples

int init: Inicio de los registros en el archivo

int n_rec: Número de Registros en el archivo

Funciones

string toRecord(string): Método para añadir el espacio necesario para que el archivo sea de longitud fija

void makeSimpleIndex(): Crea el Índice Simple

string getElement(QDomElement, QString, QString): Obtiene un Atributo de un archivo xml

void on_actionNuevo_triggered(): Crea un nuevo archivo

void on_actionCrear_triggered(): Crea un campo

void on_actionSalir_triggered(): Sale del programa

void on_actionListar_triggered():Lista los campos existentes

void on_actionModificar_triggered(): Modifica los campos existentes

void on_actionGuardar_triggered(): Guarda el trabajo realizado hasta el momento

void on_actionCerrar_triggered(): Cierra el archivo abierto

void on_actionAbrir_triggered(): Abre un archivo

void on_actionIntroducir_triggered(): Crea un registro

void on_actionBuscar_triggered(): Busca un registro

void on_actionBorrar_triggered(): Elimina un registro

void on_actionListar_2_triggered(): Lista los registros

void on_actionCruzar_triggered(): Intersección de dos archivos

void on_actionImprimir_triggered(): imprime en pdf los registros del archivo

void on_actionCrear_Indices_Simples_triggered(): Crea los índices simples

void on_actionReindexar_triggered(): Vuelve a crear los índices

void on_actionExportar_XML_triggered(): Exporta el archivo en xml

void on_actionExportar_Json_triggered(): Exporta el archivo en json

void on_actionImportar_XML_triggered(): Importa el archivo en xml

void on_actionImportal_Json_triggered(): Importa el archivo en json

BTree

Atributos

BNode* root: raíz del árbol

Funciones

BTree(): constructor de la clase

~BTree(): destructor de la clase

void insert (string): método de inserción

void remove (string): método de eliminación de llaves

bool search_Key (BNode*,string): busca la llave a partir de un nodo y retorna si existe

BNode* search_RemoveNode(BNode*, string): busca el nodo del que se removera la llave

BNode* getRoot(): retorna la raíz

BNode* search_InsertNode (BNode*,string): busca el nodo donde se insertara la llave

void split_promote (BNode*): realiza el procedimiento de split promote en los nodos

void print(): imprime el árbol

string swap(BNode**, string): realiza el intercambio de llaves en la eliminación

void rebalance(BNode**): realiza el método de redistribución

BNode* swapNode(BNode**, string k): retorna el nodo en que se hizo el intercambio en la eliminación

void concatenate(BNode**): método de concatenación

BNode

Atributos

BNode* parent: padre del nodo

BNode* pointers[5]: arreglo de hijos

string keys[4]: llaves

int keycount: contador de llaves

bool isleaf: dice si es hoja

Funciones

BNode(): constructor de la clase

~BNode(): destructor de la clase

void addKey(string): agrega una llave al nodo

bool searchKey(string): busca una llave en el nodo

void setParent (BNode*): asigna el padre del nodo

int getKeyPosition(string): retorna la posición de la llave en el arreglo

void setPointers(BNode**, BNode**, int): asigna los apuntadores de la izquierda y la derecha del nodo nuevo

void setChildren(BNode**, int): asigna un nodo hijo en la posición del arreglo correspondiente

BNode* getParent (): retorna el padre del nodo

BNode* getChildren(int): retorna el hijo en la posición indicada

int getChildrenPosition(BNode**): retorna la posición del nodo hijo

BNode* getPosition(string): retorna el puntero de la llave especificada en inorder

void setKeyCount (int): asigna el contador de llaves

int getKeyCount(): retorna el contador de llaves en el arreglo

void setIsLeaf(bool): asigna si es hoja

bool getIsLeaf(): retorna si es hoja

string getKey(int): retorna la llave en la posición indicada

void setKey(int, string): asigna la llave en la posición indicada

string toString(): retorna el toString

void print(): imprime la información del árbol y los subárboles

void sortKeys(): arregla las llaves

void removeKey(string k): remueve la llave del arreglo

void removeChildren(int): remueve un hijo de la posición indicada

Manual de Usuario

En esta sección podemos ver paso por paso la interacción que se tiene del usuario con el programa. Ingresamos al programa y lo primero que vamos a ver es una interfaz sencilla que tiene una barra de menú en la parte superior. Las opciones a escoger son:

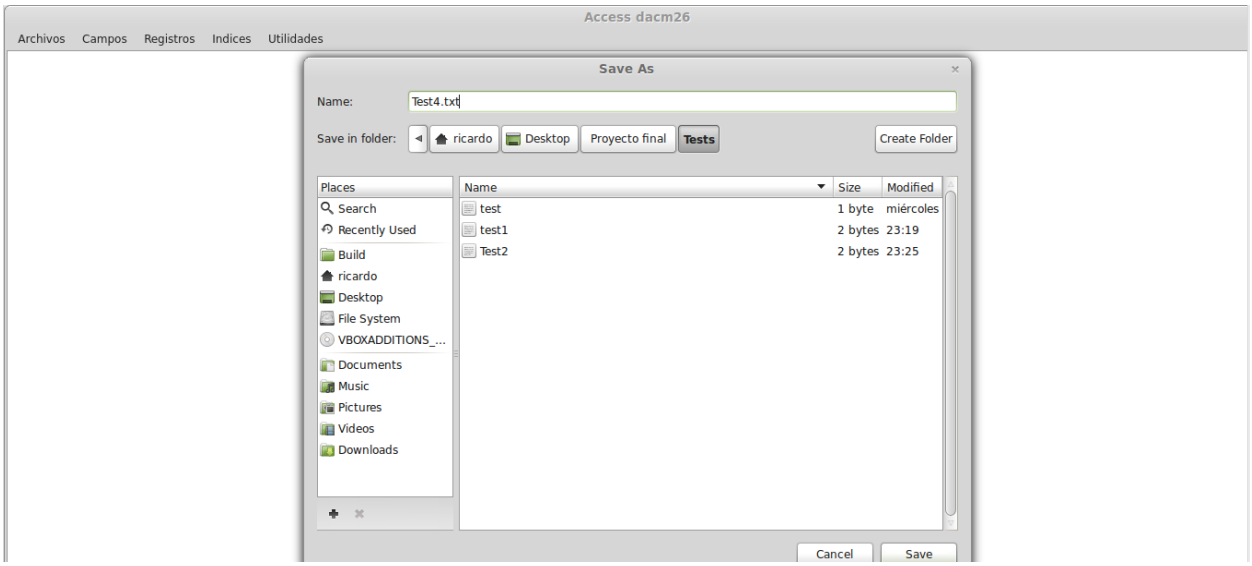
- Archivos
- Campos
- Registros
- Índices
- Utilidades

Al acceder a la viñeta de archivos se presenta la siguiente ventana:

Aquí se encuentran las opciones de:

- Crear nuevo archivo
- Abrir nuevo archivo
- Guardar un archivo abierto
- Imprimir en pdf el archivo
- Cerrar el archivo abierto
- Salir del sistema

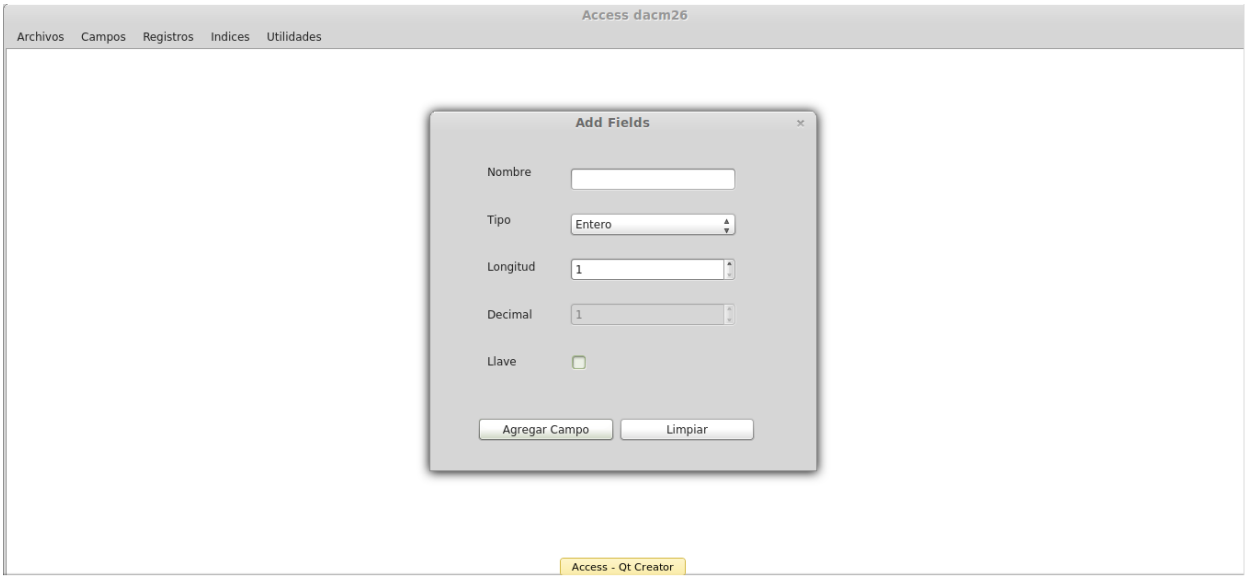
Al crear un nuevo archivo, se nos desplegara una ventana de dialogo para indicar el lugar donde se guardará el nuevo archivo.



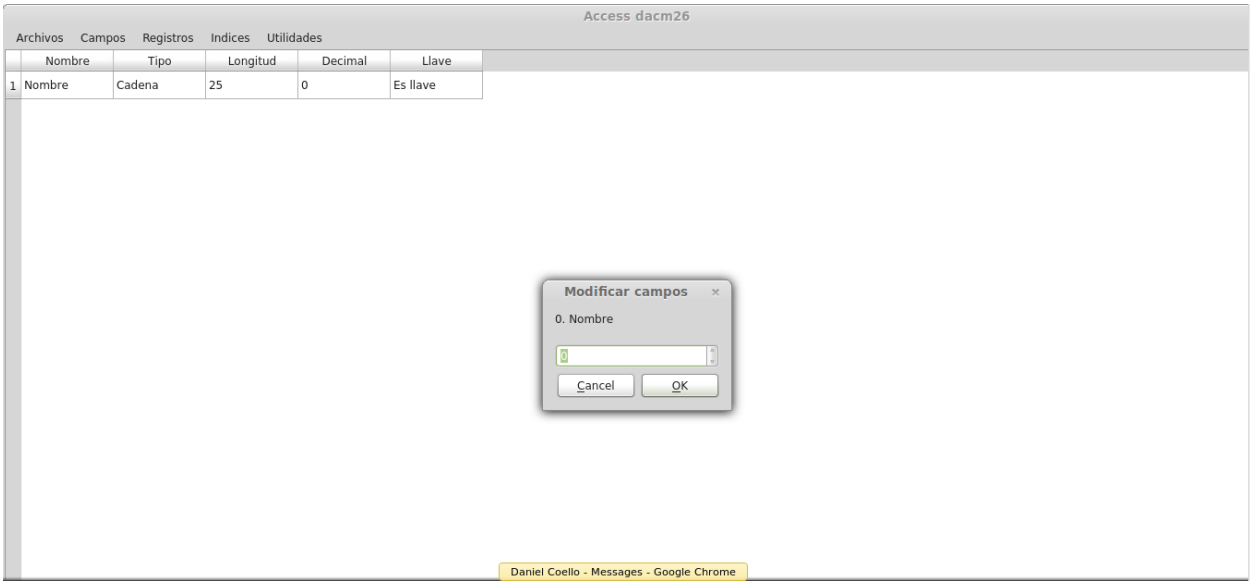
Una vez creado el archivo, podemos proceder a crear campos para ingresarle. Al ingresar a la viñeta de campos podemos ver estas opciones:

- Crear campo
- Modificar campo
- Listar campo

Cuando ingresamos a la ventanilla de crear un nuevo campo, se nos presenta un cuadro de diálogo el cual nos da la facilidad de asignarle al nuevo campo, un nombre, longitud, tipo de información y si es llave primaria o no.



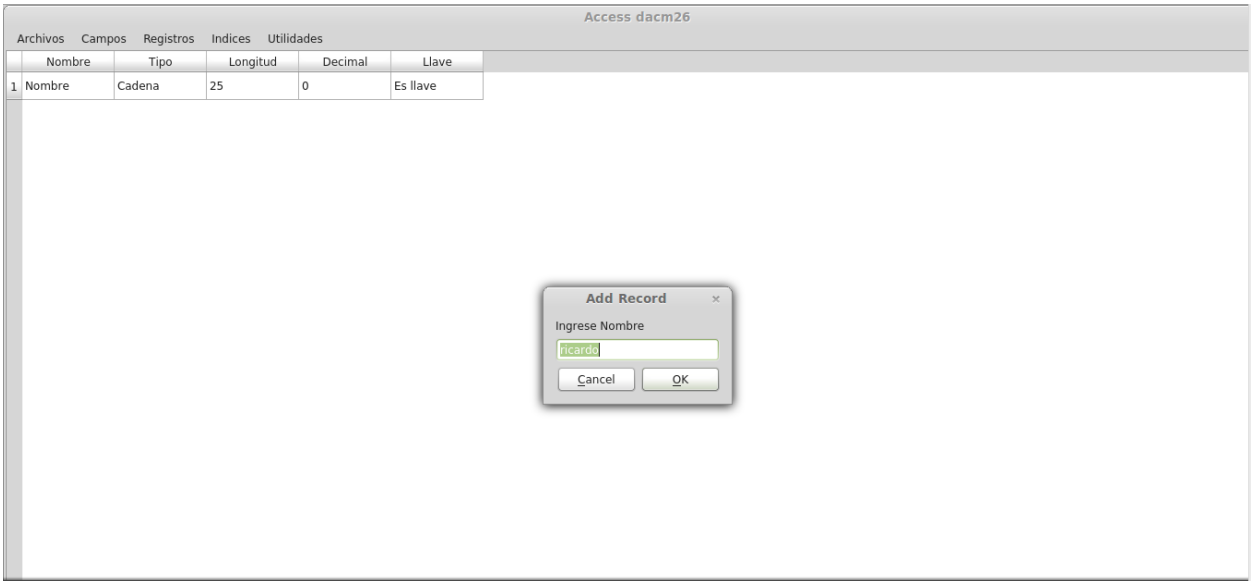
Para modificar un campo, se nos presenta un cuadro de diálogo que nos permite escoger uno de los campos para modificar.



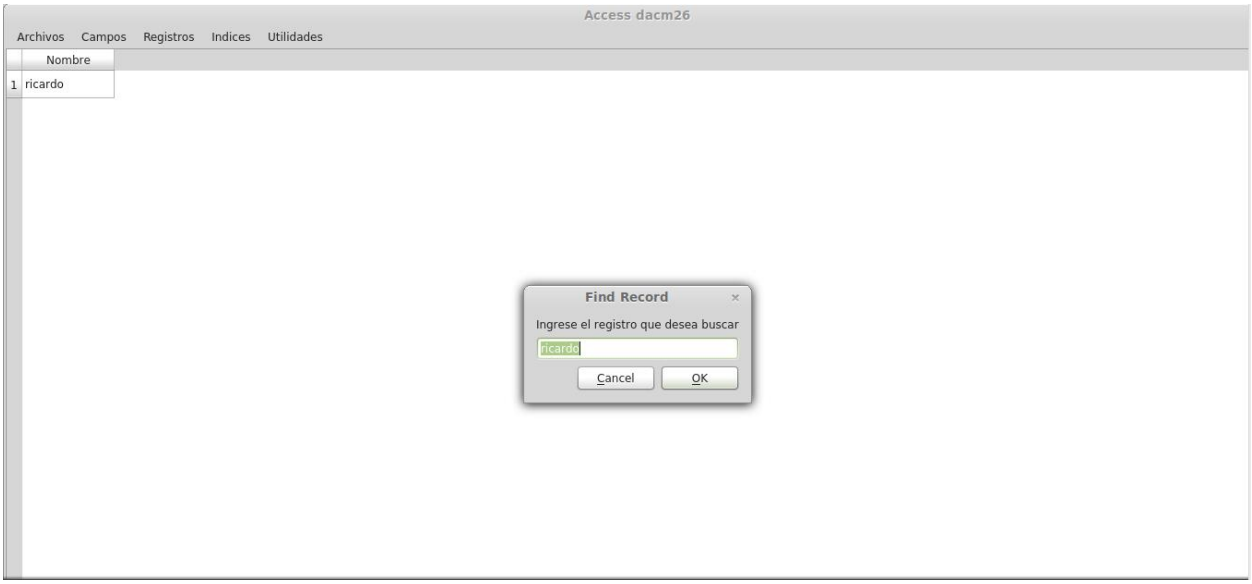
Después que hemos creados cuantos campos sean necesarios, podemos comenzar a insertar registros. Nos dirigimos a la ventana de Registro y se nos presentan las siguientes opciones:

- Introducir
- Buscar
- Borrar
- Listar

Al acceder a introducir, se nos presenta un cuadro de dialogo donde se nos pide que ingresemos la información campo por campo, según el tipo de información que se escogió.

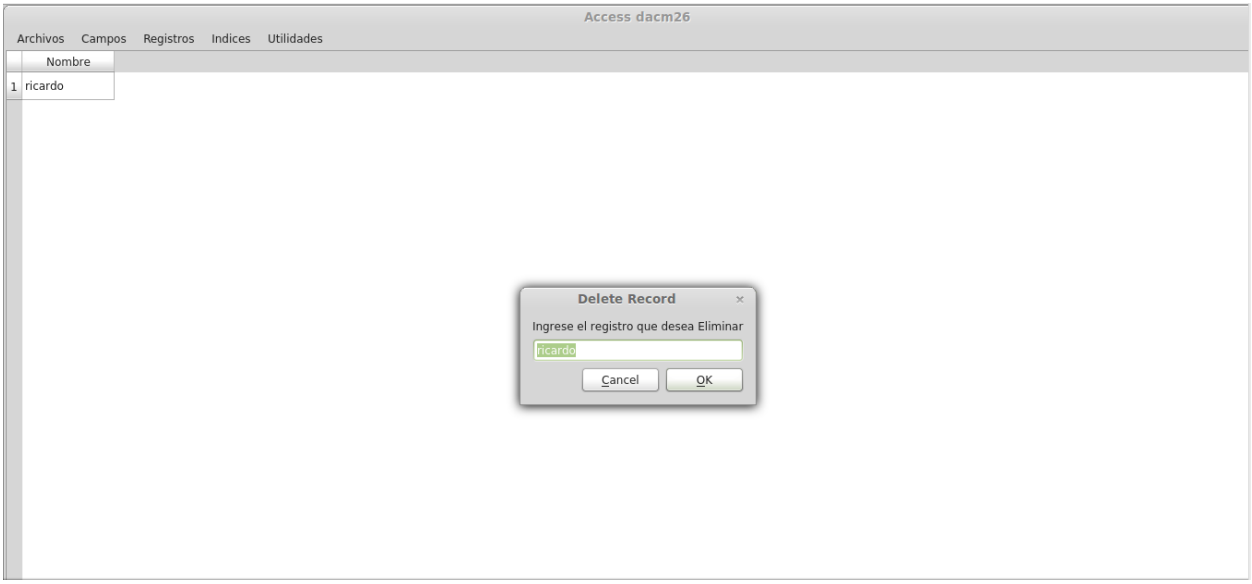


Para buscar un registro, accedemos a la opción en el menú superior y se nos presenta la siguiente ventana de diálogo.



Como se puede ver, se pide que se ingrese un valor que corresponde a la llave principal del registro. A través de este valor, se determinará si existe el registro o no.

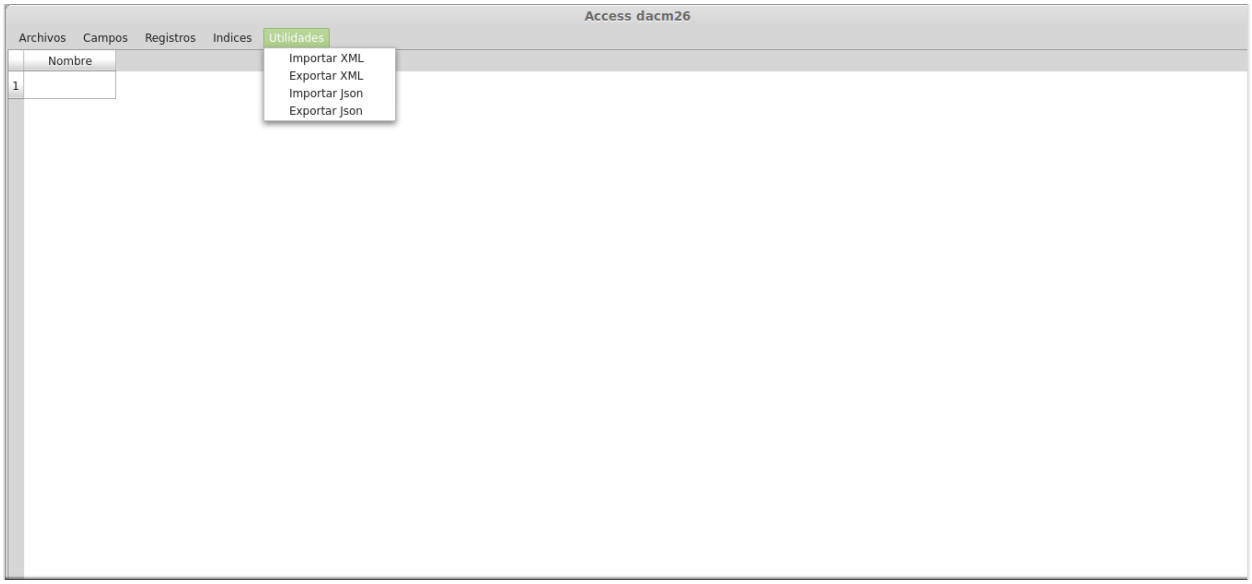
Una vez que hemos ingresado campos al sistema, se podrán visualizar en la tabla que se encuentra localizada en el centro de la pantalla. Si quisiéramos eliminar algún registro que ha sido creado, simplemente nos vamos a la opción en el menú superior y se nos pedirá ingresar un valor del registro según su llave primaria.



El programa nos da la facilidad de poder generar índices a partir de los registros ya creados. Estos índices serán utilizados para verificar y buscar registros a través de su llave primaria y su posición en el archivo. Usando las opciones en el menú localizado en la ventana superior, podemos crear índices simples y montarlos en el árbol b.

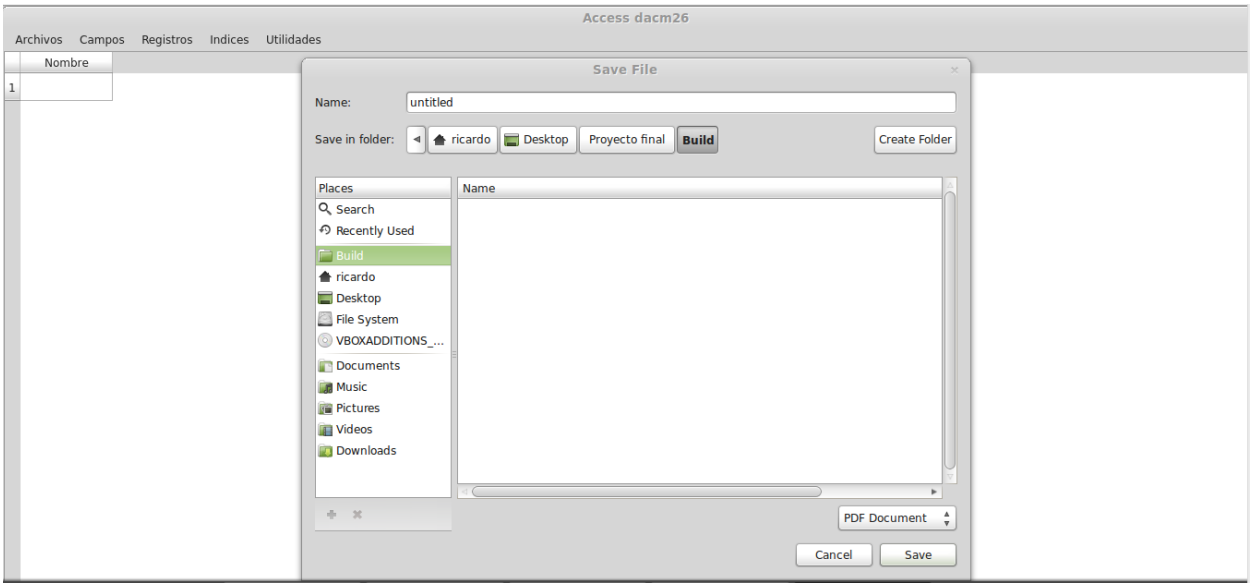
Por último, tenemos una opción de utilidades en el menú. En esta opción se nos presentan 4 elecciones:

- Importar XML
- Exportar XML
- Importar JSON
- Exportar JSON



En esta ventanilla, podemos exportar nuestro trabajo a un documento en formato de XML o JSON. También podemos importar a nuestro archivo, un documento en formato de XML o JSON e integrarlo a nuestro programa.

Una vez que hemos terminado de trabajar en nuestro archivo, podemos acceder a la opción de Archivo en el menú y darle guardar. Esto guardará todos los cambios realizados en nuestro programa. Si el usuario gusta, también puede imprimir a formato pdf todo el archivo de registro.



Experiencia en GitHub

Github es una gran herramienta para nosotros, ya que con ella se hace más fácil trabajar en grupo e intercambiar código. Pero no solo en eso nos ayudó, ya que tuvimos varios problemas al momento de probar nuestro programa y gracias a Github pudimos obtener la última versión “Estable” que teníamos y con esto nos ahorramos una gran cantidad de tiempo, porque no perdemos tiempo buscando los errores de nuestro programa. En nuestra opinión se debería enseñar a cómo utilizar Github desde programación I, ya sea con talleres o hacer que el estudiante investigue como utilizarlo, porque en si no es difícil aprender a usar Github.

Conclusiones

En conclusión fue una buena experiencia realizar este proyecto, ya que es un desafío muy interesante y muy difícil de programar, porque se practica absolutamente todos los conocimientos adquiridos en la clase. Aprendimos a tomar en cuenta aún más la memoria y el rendimiento de nuestras funciones, además de aprender a reducir los accesos a discos.