UNIVERSIDAD SAN CARLOS DE GUATEMALA
CENTRO UNIVERSITARIO DE OCCIDENTE
DIVISIÓN CIENCIAS DE LA INGENIERIA
INGENIERIA CIENCIAS Y SISTEMAS
ESTRUCTURA DE DATOS
ING. OLIVER ERNESTO SIERRA
AUX. DANIEL ALBERTO GONZALEZ

MANUAL TÉCNICO

SEGUNDO PROYECTO “GUATEMALA DB’S”

201730919 BRYAN RENÉ GÓMEZ GÓMEZ

Métodos y Clases

Clase main:

void querys(string, ManejadorCadenas);
Método que reconoce la sintaxis de una query e interpreta cada una para realizar la acción solocitada.

void menu();
Método que muestra el menú principal con cada una de sus opciones posteriormente manda a llamar al método encargado de cada uno.

void menuReportes();
Método que muestra el menú de reportes con cada una de sus opciones posteriormente manda a llamar al método encargado de cada uno.

void dbActual();
Método que selecciona una DB.

void crearDB();
Método que crea una DB y lo agrega a la lista de DB's.

void cambiarDB();
Método para cambiar de DB's.

void queryCrear(Create &crear);
Método manejador de la query Create.

void queryInsert(Insert &insert);
Método manejador de la query Insert.

void querySelect(Select &select);
Método manejador de la query Select.

void realizarSelect(Select &, NodoTabla *, NodoColumna *, ListaCadena &);
Método encargado de realizar la instrucción select, mediante cada una de una de sus condiciones.

void cantidadFilasDeUnMismoTipo();
Método encargado de revisar la cantidad de filas de un mismo tipo en una tabla de la DB actual.

void imprimirGraphviz();
Método encargado de escribir en un archivo con la estructura de una DB y poder visulizar mediante graphviz para que posteriormente sea compilado en la terminal mediante el comando: **dot -Tpng archivo.dot -o imagen.png.**

int colisionLineal(int, int);
Método manejador de las colisiones al momento de realizar una inserción en la tabla hash de una columna.

bool comprobarElTipoDatoColumna(int, string);
Método que comprueba el tipo de dato de una columna: String, Char, Int o Double.

bool esCaracter(string);
Método que comprueba si una cadena es un carácter.

bool esDecimal(string);
Método que comprueba si una cadena es un decimal.

bool esNumero(string);
Método que comprueba si una cadena es un número.

void imprimirSelect(ListaCadena &);
Método que imprime en consola la instrucción realizada select, cada uno de los datos seleccionados.

void limpiarCadena(int, ListaCadena &, ListaCadena);
Método encargado de limpiar una cadena de espacios

void ingresarAlLog(string concatenar);
Método que escribe a una archivo log el registro de las acciones dentro del programa.

void cantidadDatosDB();
Método que verifica la cantidad de datos en cada una de las DB’s y las muestra en consola.

Clase Arbol:

Arbol(); y Arbol(const Arbol& orig);
Constructores.

virtual ~Arbol();
Destructor.

bool ingresarDatosHoja(string, string, int);
Función que ingresa una nueva hoja al arbol regresa verdadero si se realizo con éxito de lo contrario false.

void insertarHoja(NodoHoja*, NodoHoja*, NodoHoja*);
Método que realiza la inserción de un nuevo nodo al árbol.

void equilibrar(NodoHoja *);
Método que equilibra el árbol después de que se realice una nueva inserción.

void rotarDD(NodoHoja *, NodoHoja *);
Método que realiza una rotación simple a la derecha.

void rotarDI(NodoHoja *, NodoHoja *);
Método que realiza una rotación derecha izquierda.

void rotarII(NodoHoja *, NodoHoja *);
Método que realiza una rotación simple a la izquierda.

void rotarID(NodoHoja *, NodoHoja *);
Método que realiza una rotación izquierda derecha.

void agregarAtributosHoja(NodoHoja *&, string, string, int);
Método que agrega los atributos a un nodo hoja.

bool comprobarExistencia(NodoHoja*, int);
Función que comprueba la existencia de un dato en el árbol.

int calcularAltura(NodoHoja*);
Función que calcula la altura del árbol luego de que se inserte una nueva hoja al árbol.

void verArbol(NodoHoja*, int);
Método que imprime el arbol en consola.

bool buscarHoja(NodoHoja *, int);
Método que busca una hoja en el arbol.

bool digito(string);
Función que comprueba si una cadena es un número.

bool decimal(string);
Función que comprueba si una cadena es un decimal.

int tipoDato(string);
Función que comprueba el tipo de dato de una cadena: String, Char, Int o Double.

void vaciarArbol(NodoHoja* recorrer);
Método que vacia un arbol.

int charANumero(char);
Función que convierte un char a número.

bool validarCondicion(string, int, string, int);
Función que valida la condición de una query select si corresponde a la columna el tipo de dato.

string graphivzArbol(NodoHoja *, int &, string);
Función que realiza la estructura graphviz del arbol y sus nodos.

string seleccionar(NodoHoja *, string, bool, int);
Función que retorna los datos de la instrucción select que están dentro del árbol.

Clase ListaCadena:

ListaCadena(); y ListaCadena(const ListaCadena& orig);
Constructores.

virtual ~ListaCadena();
Destructor.

Nodo* GetNodo(int);
Función que obtiene un nodo de la lista cadena mediante su indice.

int size();
Función que determina el tamaño de la lista cadena.

void insertar(string, int);
Método que inserta un nuevo nodo a la lista cadena.

void eliminar(Nodo *&);
Método que elimina un nodo de la lista.

void desplegarLista();
Método que imprime en consola cada uno de los datos guardados en la lista.

Clase ListaColumnas:

ListaColumnas(); y ListaColumnas(const ListaColumnas& orig);
Constructor.

virtual ~ListaColumnas();
Destructor.

NodoColumna* GetNodo(int);
Función que obtiene un nodo columna de la lista columna mediante su indice.

NodoColumna* buscarColumna(string);
Función que busca una cadena de la lista columna mediante su nombre.

int size();
Función que determina el tamaño de la lista cadena.

int cantidadDatos();
Función que determina la cantidad de datos que tiene cada una de las columnas.

int cantidadFilas(int);
Función que determina la cantidad de filas de un mismo dato que tiene cada una de las columnas.

string graphvizColumnas(int &, string);
Función que realiza la estructura graphviz de cada columna con cada tabla hash.

void insertarColumna(string, int);
Método que ingresa una nueva columna a la lista.

void eliminarColumna(NodoColumna*&);
Método que elimina un nodo columna de la lista.

void desplegarColumnas();
Método que imprime en consola cada uno de las columnas guardadas en la lista.

Clase ListaDeDB:

ListaDeDB(); y ListaDeDB(const ListaDeDB& orig);
Constructores.

virtual ~ListaDeDB();
Destructor.

NodoDB* GetNodo(int);
Función que obtiene un nodo db de la lista db's mediante su indice.

NodoDB* buscarDB(string);
Función que busca base de datos en la lista db's mediante su nombre.

int size();
Función que determina el tamaño de la lista db's.

void insertarDB(string);
Método que ingresa una nueva DB en la lista.

void eliminarDB(NodoDB*&);
Método que elimina un nodo db de la lista.

void desplegarDB();
Método que imprime en consola cada uno de las db's guardadas en la lista.

Clase ListaHash

ListaHash(); y ListaHash(const ListaHash& orig);
Constructor.

virtual ~ListaHash();
Destructor.

NodoArbolAVL *GetNodo(int);
Función que obtiene un nodo arbol avl de la lista hash mediante su indice.

int size();
Función que determina el tamaño de la lista hash.

int cantidadDatos();
Función que determina la cantidad de datos que contiene la lista hash.

string graphivzHash(int &);
Función que realiza la estructura graphviz de cada lista hash con cada árbol avl.

string datos(string, string, bool, int);
Función que obtiene los datos de las lista hash y cada una de los arboles.

void realizarReHashing();
Método que realiza el rehashing de la tabla si sobre pasa el factor de carga = 60%.

void insertarNuevaTupla();
Método que inserta una nueva tupla a la tabla hash.

void eliminarTupla(NodoArbolAVL*&);
Método que elimina una tupla de la tabla hash.

void desplegarLista();
Método que imprime en consola cada uno de las las tuplas guardadas en la tabla hash.

Clase ListaTablas

ListaTablas(); y ListaTablas(const ListaTablas& orig);
Constructor.

virtual ~ListaTablas();
Destructor.

NodoTabla* GetNodo(int);
Función que obtiene un nodo tabla de la lista tablas mediante su indice.

NodoTabla* buscarTabla(string);
Función que buscar una tabla en la lista tablas mediante su nombre.

int size();
Función que determina el tamaño de la lista tablas.

void insertarTabla(string);
Método que inserta una nueva tabla a la lista de tablas.

void eliminarTabla(NodoTabla*&);
Método que elimina un nodo tabla en la lista de tablas.

void desplegarTabla();
Método que imprime en consola cada uno de las las tablas guardadas en la lista.

int cantidadDatos();
Función que determina la cantidad de datos en todas las tablas.

int cantidadColumnas();
Función que determina la cantidad de columnas en todas las tablas.

string graphvizTabla(int &, string, bool, int);
Función que realiza la estructura graphviz de cada tabla con cada una de su tabla hash.

Clase ManejadorCadenas

ManejadorCadenas(); y ManejadorCadenas(const ManejadorCadenas& orig);
Constructores.

virtual ~ManejadorCadenas();
Destructor.

int cuenta(string, const char, int&);
Función que cuenta la cantidad de caracteres de un mismo tipo que contiene una cadena.

void split(string, char , ListaCadena& , int&);
Método que separa una cadena por un carácter definido.

Clase ManejadorHash:

ManejadorHash(); y ManejadorHash(const ManejadorHash& orig);
Constructor.

virtual ~ManejadorHash();
Destructor.

bool digito(string);
Función que comprueba si una cadena es un número.

bool decimal(string);
Función que comprueba si una cadena es un decimal.

int charANumero(char);
Función que convierte un carácter a un número.

long long cadenaANumero(string valor);
Función que convierte una cadena a un número.

int funcionHash(long long, int);
Función hash que determina el indice de donde sera insertado un dato en la tabla hash.

int llave(int, string, int);
Función que obtiene la llave de un dato.

int dato(int, string);
Función que determina el tipo de dato de una cadena.

Clase ManejadorQuery

ManejadorQuery(); y ManejadorQuery(const ManejadorQuery& orig);
Constructor.

virtual ~ManejadorQuery();
Destructor.

int tipoDato(string);
Función que determina el tipo de dato de una columna de la instrucción create.

int condicion(string);
Función que determina el tipo de condición de la instrucción select.

Create create(string, ManejadorCadenas);
Método que realiza el análisis de una instrucción create y construye un objeto create mediante las condiciones que presenta la instrucción.

void remplazar(string &, char, char);
Método que remplaza un carácter por otro en una cadena.

void select(string, ManejadorCadenas, Select &);
Método que realiza el análisis de una instrucción select y construye un objeto seleccionar mediante las condiciones que presenta la instrucción.

void insertarALista(int, ListaCadena &, ListaCadena&);
Método que inserta una cadena a una lista.

void insert(string, ManejadorCadenas mc, Insert &);
Método que realiza el análisis de una instrucción insert y construye un objeto insert mediante las condiciones que presenta la instrucción.

void limpiarCadenaDeEspacios(int, string &);
Método que limpia una cadena de espacios al inicio.

void limpiarCadenaDeEspaciosFinales(string &);
Método que limpia una cadena de espacios al fin.

ESTRUCTURAS:
- **Arbol**
- **Lista Simple Cadena**
- **Lista Simple Columna**
- **Lista Simple DB’s**
- **Lista Simple Hash**
- **Lista Simple Tabla**

TYPEDEF STRUCTS:
- **NodoHoja**
- **Nodo**
- **NodoColumna**
- **NodoDB**
- **NodoArbolAVL**
- **NodoTabla**

OBJETOS:
- **Create**
- **Insert**
- **Select**

Nota: Cada uno con sus respectivo contructor, getter's y setter's.

Librerias Requeridas:

<cstdlib>
<ctype.h>
<stdlib.h>
<iostream>
<stdio.h>
<ctime>
<time.h>
<cstring>
<string>
<sstream>
<fstream>

ARCHIVO MAKE:

Compila cada una de las clases y crea un archivo ejecutable para poder iniciarlo en un terminal.

Proyecto_DB: main.o Arbol.o Create.o Insert.o ListaCadena.o ListaColumnas.o ListaDeDB.o ListaHash.o ListaTablas.o ManejadorCadenas.o ManejadorHash.o ManejadorQuery.o Select.o

g++ main.o Arbol.o Create.o Insert.o ListaCadena.o ListaColumnas.o ListaDeDB.o ListaHash.o ListaTablas.o ManejadorCadenas.o ManejadorHash.o ManejadorQuery.o Select.o -o Proyecto_DB

./Proyecto_DB

Compila cada una de las clases: archivos .cpp y .h

main.o: main.cpp

g++ -c main.cpp

Arbol.o: Arbol.cpp Arbol.h

g++ -c Arbol.cpp

Create.o: Create.cpp Create.h

g++ -c Create.cpp

Insert.o: Insert.cpp Insert.h

g++ -c Insert.cpp

ListaCadena.o: ListaCadena.cpp ListaCadena.h

g++ -c ListaCadena.cpp

ListaColumnas.o: ListaColumnas.cpp ListaColumnas.h

g++ -c ListaColumnas.cpp

ListaDeDB.o: ListaDeDB.cpp ListaDeDB.h
g++ -c ListaDeDB.cpp

ListaHash.o: ListaHash.cpp ListaHash.h
g++ -c ListaHash.cpp

ListaTablas.o: ListaTablas.cpp ListaTablas.h
g++ -c ListaTablas.cpp

ManejadorCadenas.o: ManejadorCadenas.cpp ManejadorCadenas.h
g++ -c ManejadorCadenas.cpp

ManejadorHash.o: ManejadorHash.cpp ManejadorHash.h
g++ -c ManejadorHash.cpp

ManejadorQuery.o: ManejadorQuery.cpp ManejadorQuery.h
g++ -c ManejadorQuery.cpp

Select.o: Select.cpp Select.h
g++ -c Select.cpp

REQUISITOS MÍNIMOS PARA PROGRAMAR EN C++ CON NETBEANS 8.2:

REQUERIMIENTOS DE HARDWARE PARA PROGRAMAR EN C++ CON NETBEANS 8.2
Para las aplicaciones generadas se debe tener un mínimo de 32MB de RAM, se recomienda que se tengan 48MB o más. Para compilar utilizando el SDK de Microsoft es un mínimo de 32MB de RAM, y para JDK 48MB mínimo. El procesador en principio no es tan crítico como la memoria RAM, pero se recomienda utilizar al menos un Pentium de 133 para compilar/ejecutar las aplicaciones.

Requerimientos de SOFTWARE
Para trabajar con el Generador Java es necesario cierto software adicional para la compilación y ejecución de las aplicaciones generadas.

(Java necesario para instalar NetBeans 8.2)

Java Development Kit - Compilador y Máquina Virtual
Para la compilación y ejecución de los programas es necesario tener el JDK. En cuanto a las versiones se recomienda en general utilizar las últimas liberadas de cada uno de ellos. En caso de que alguna versión sea menos estable que su versión anterior, se avisara.

En Windows se requieren las versiones siguientes de .NET Framework:
SQL Server 2008 en Windows Server 2003 64 bits IA64: .NET Framework 2.0 SP2SQL Server Express: .NET Framework 2.0 SP2. Todas las demás ediciones de SQL Server 2008: .NET

Framework 3.5 SP1, etc.

IDE NetBeans 8.2 requisitos mínimos:

Microsoft Windows Vista SP1 / Windows 7 Professional:
Procesador: Varel Pentium III 800MHz o equivalente
Memoria: 512 MB
Espacio en disco: 750 MB de espacio libre en disco
Ubuntu 9.10:
Procesador: Varel Pentium III 800MHz o
equivalente Memoria: 512 MB
Espacio en disco: 650 MB de espacio libre en disco

Macvarosh OS X 10.7
Varel: Procesador: Varel
Dual-Core Memoria: 2 GB
Espacio en disco: 650 MB de espacio libre en disco

Microsoft Windows 7 Professional / Windows 8 / Windows 8.2 / Windows 10:
Procesador: Varel Core i5 o
equivalente Memoria: 2 GB 32 bits, 4
GB 64 bits
Espacio en disco: 1,5 GB de espacio libre en disco

Ubuntu 15.04 y Superior:
Procesador: Varel Core i5 o
equivalente Memoria: 2 GB 32 bits, 4
GB 64 bits
Espacio en disco: 1,5 GB de espacio libre en disco

OS X 10.10 Varel:
Procesador: Varel Dual-
Core Memoria: 4 GB
Espacio en disco: 1,5 GB de espacio libre en disco