Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Lenguajes Formales y de Programacion
Sección A-
Inga. Vivian Damaris Campos Gonzales
Aux. Mario Josué Solis Solórzano
Primer semestre 2023

MANUAL DE USUARIO

PROYECTO 2: DISEÑO Y CREACION DE SENTENCIAS DE BASES DE DATOS MEDIANTE UN ANALIZADOR LEXICO Y SINTACTICO

Nombre:	Eduardo Misael Lopez Avila	Registro académico:	202100147
Fecha:	30/04/2022	Sección:	A-

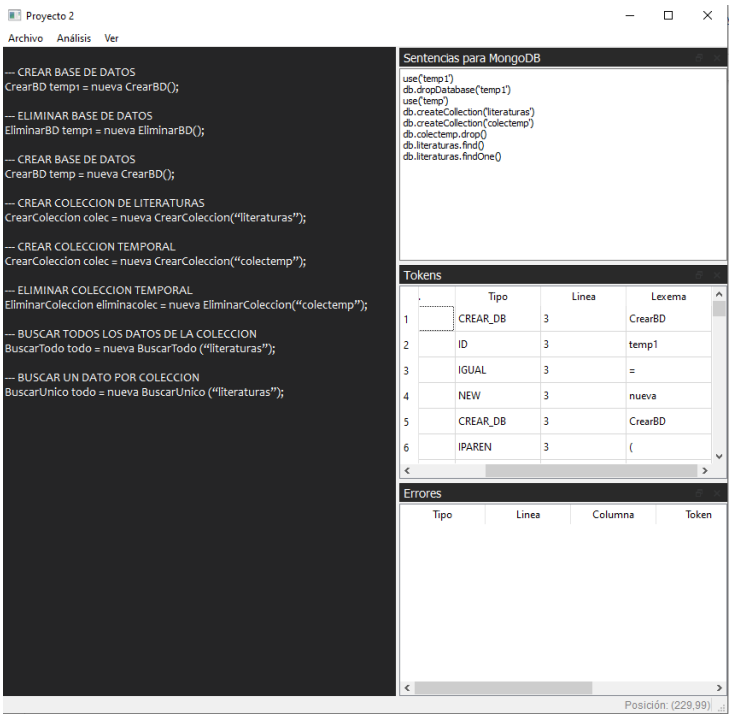
1. Descripción

El proyecto consiste en la elaboración de una herramienta que permita el diseño y creación de sentencias de bases de datos no relacionales de una forma sencilla. La aplicación tendrá un área de edición de código y un área de visualización de la sentencia final generada. Cuando ya se cuente con las sentencias creadas inicialmente, se procederá a realizar la compilación respectiva lo que generar las sentencias de MongoDB que serán mostradas en el espacio de resultados que posteriormente se podrán aplicar a un entorno adecuado a MongoDb.

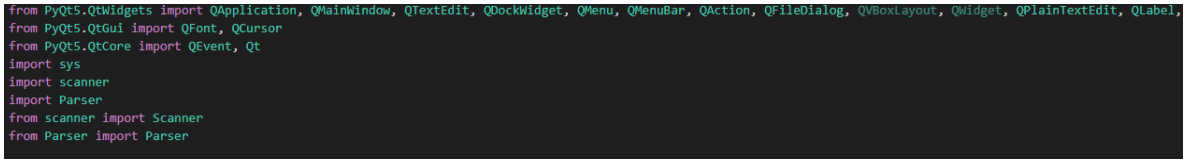
2. Aplicación de lectura de archivo de carga.

2.1. Vista general de la aplicación

Se inicializa la aplicación.



2.2. Librerias en uso



```
from PyQt5.QtWidgets import QApplication, QMainWindow, QTextEdit, QDockWidget, QMenu, QMenuBar, QAction, QFileDialog, QVBoxLayout, QWidget, QPlainTextEdit, QLabel,
from PyQt5.QtGui import QFont, QCursor
from PyQt5.QtCore import QEvent, Qt
import sys
import scanner
import Parser
from scanner import Scanner
from Parser import Parser
```

PyQt5: es una librería de Python que provee un conjunto de herramientas para la creación de aplicaciones gráficas con una interfaz de usuario (UI) avanzada. Utiliza el popular framework Qt, escrito en C++, para proveer una gran cantidad de funcionalidades para la creación de GUIs, incluyendo ventanas, botones, cajas de texto, menús, y muchos otros elementos. PyQt5 es una opción popular para la creación de aplicaciones de escritorio y su uso es muy extendido en la industria.

Sys: es una librería de Python que proporciona acceso al intérprete de Python y a las variables y funciones que están dentro de él. Contiene funciones y variables que se utilizan comúnmente para interactuar con el sistema operativo, como sys.exit() que permite salir de un programa de Python, o sys.argv que proporciona acceso a los argumentos de línea de comandos que se pasan al script.

2.3. Interfaz grafica

```
class ventanaGrafica(QMainWindow):
    def __init__(self):
        super().__init__()

        # Ventana principal
        self.setWindowTitle('Proyecto 2')
        self.setGeometry(100, 100, 800, 800)

        # Agregar widgets y configuraciones adicionales para la interfaz de usuario
        self.iniciarUI()
        QApplication.instance().installEventFilter(self)  #Se inicializa la instancia para el cursor X y Y

    #Parametros de la interfaz grafica
    def iniciarUI(self):
```

Se crea la clase ventanaGrafica que se inicializa en la ejecución del programa, dentro de ella se definen los widgets a utilizar de PyQt5, esto para que sea mas agradable para el usuario.

```
def iniciarUI(self):
    #Area de editor de texto
    self.cajaTexto = QPlainTextEdit(self)
    self.cajaTexto.setFont(QFont("Candara", 12))
    self.cajaTexto.setStyleSheet('''
        QPlainTextEdit {
            background-color: #252526;
            color: #FFFFFF;
            border: none;
            font-size: 14px;
        }
    ''')
    self.setCentralWidget(self.cajaTexto)

    #Label del cursor en X y Y
    self.label_posicion = QLabel(self)
    self.label_posicion.setAlignment(Qt.AlignRight | Qt.AlignVCenter)
    self.label_posicion.setStyleSheet('font-size: 12px; font-family: Arial; color: gray;')
    self.statusBar().addPermanentWidget(self.label_posicion)
    self.label_posicion.setText('Posición: (0,0)')


    # Area de visualización de sentencias
    self.visorTraductor = QTextEdit(self)
    self.visorTraductor.setReadOnly(True)
    self.widgetTraductor = QDockWidget("Sentencias para MongoDB", self)
    self.widgetTraductor.setStyleSheet('''
        QDockWidget {
```

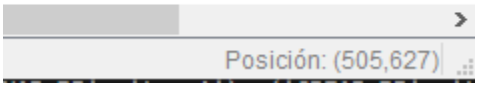
Dentro de la función iniciarUI, se definen todos los widgets que se pueden visualizar al ejecutar la aplicación, se definen que parámetros tienen, dimensiones, fuentes, colores y cualquier capa de personalización que se pueda indicar dentro de cada tipo de Widget

2.4. Cursor en X y Y

```python
def actualizar_posicion(self, event):
    if event.type() == QEvent.MouseMove:
        cursor = QCursor()
        pos = cursor.pos()
        if self.rect().contains(self.mapFromGlobal(pos)):
            x = pos.x() - self.geometry().x()
            y = pos.y() - self.geometry().y()
            self.label_posicion.setText(f'Posición: ({x},{y})')

def eventFilter(self, source, event):
    if event.type() == QEvent.MouseMove:
        if self.rect().contains(self.mapFromGlobal(event.pos())):
            self.actualizar_posicion(event)
    return super().eventFilter(source, event)
```

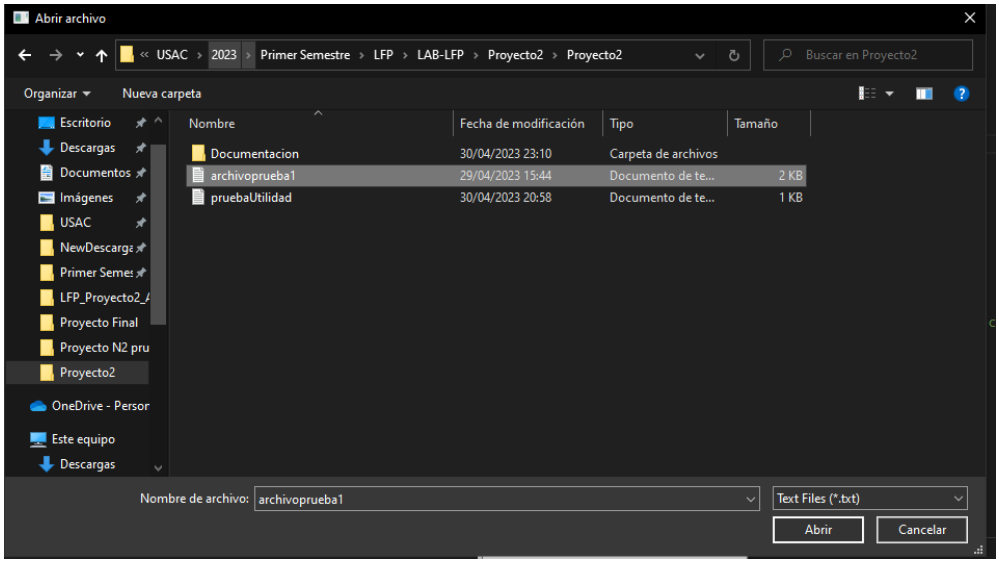
Se define la función que muestra la posición X y Y del cursor, se actualiza constantemente según la posición actual del cursor dentro de la ventana de la aplicación. Esta se puede observar en la parte inferior derecha de la ventana.



2.5. Abrir

```python
def abrirArchivo(self):
    options = QFileDialog.Options()
    file_name, _ = QFileDialog.getOpenFileName(self, "Abrir archivo", "", "Text Files (*.txt);;All Files (*)", options=options)
    if file_name:
        with open(file_name, 'r', encoding='utf-8') as file:
            self.cajaTexto.setPlainText(file.read())
            global archivo_seleccionado
            archivo_seleccionado = file_name #Se guardar el directorio exacto del archivo
```

Se debe ingresar una carga de datos de un archivo, el cual contiene los datos con una determinada estructura para operar. Para esto, se debe ingresar la opción ”Archivo” el cual abrirá una nueva ventana, donde se debe seleccionar la opción “Abrir”, para seleccionar el archivo a cargar, se debe presionar el botón “Abrir” de la ventana emergente y se cargara el archivo a la caja de texto.



Proyecto 2

Archivo Análisis Ver

```
/*
                ARCHIVO DE PRUEBAS
                CON COMENTARIOS
*/


--- CREAR BASE DE DATOS
CrearBD temp1 = nueva CrearBD();

--- ELIMINAR BASE DE DATOS
EliminarBD temp1 = nueva EliminarBD();

/*
                BASE DE DATOS DE  LITERATURAS
*/

--- CREAR BASE DE DATOS
CrearBD temp = nueva CrearBD();

--- CREAR COLECCION DE LITERATURAS
CrearColeccion colec = nueva CrearColeccion("literaturas");

--- CREAR COLECCION TEMPORAL
CrearColeccion colec = nueva CrearColeccion("colectemp");

--- ELIMINAR COLECCION TEMPORAL
EliminarColeccion eliminacolec = nueva EliminarColeccion("colectemp");
```

2.6. Funciones de la aplicación

2.6.1. Guardar Archivo

```
def guardarArchivo(self):
    global archivo_seleccionado
    if not self.cajaTexto.document().isModified():
        return
    if archivo_seleccionado:
        with open(archivo_seleccionado, 'w', encoding='utf-8') as file:
            file.write(self.cajaTexto.toPlainText())
```

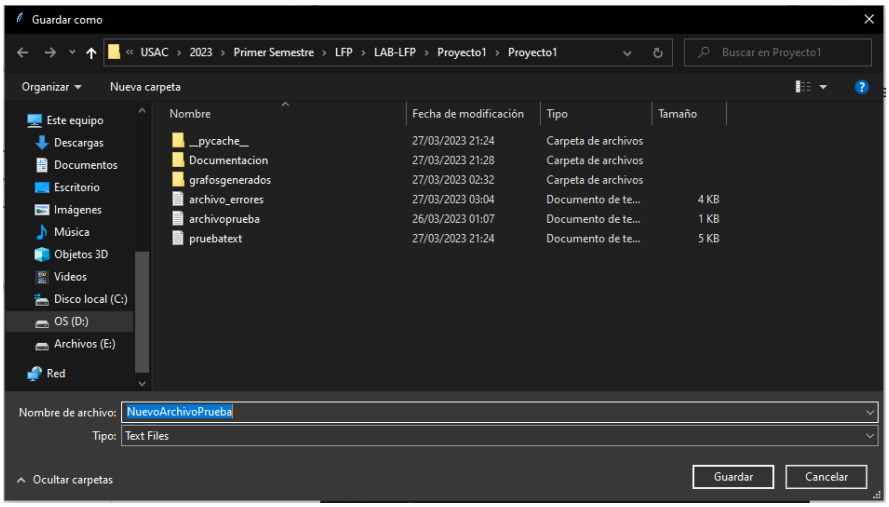
Sobrescribe el archivo que se seleccionó anteriormente por el nuevo editado desde la caja de texto de la ventana de funciones, se encodea como utf-8 para que pueda leerlo el sistema.

2.6.2. Guardar como

```
def guardarArchivoComo(self):
    options = QFileDialog.Options()
    file_name, _ = QFileDialog.getSaveFileName(self, "Guardar archivo como", "", "Text Files (*.txt);;All Files (*)", options=options)

    if file_name:
        with open(file_name, 'w', encoding='utf-8') as file:
            file.write(self.cajaTexto.toPlainText())
```

Guarda el archivo con un nombre nuevo (o el mismo) indicado por el usuario en cualquier directorio existente dentro del sistema se encodea como utf-8 para que pueda leerlo el sistema.



2.6.3. Análisis -> Generar sentencias MongoDB

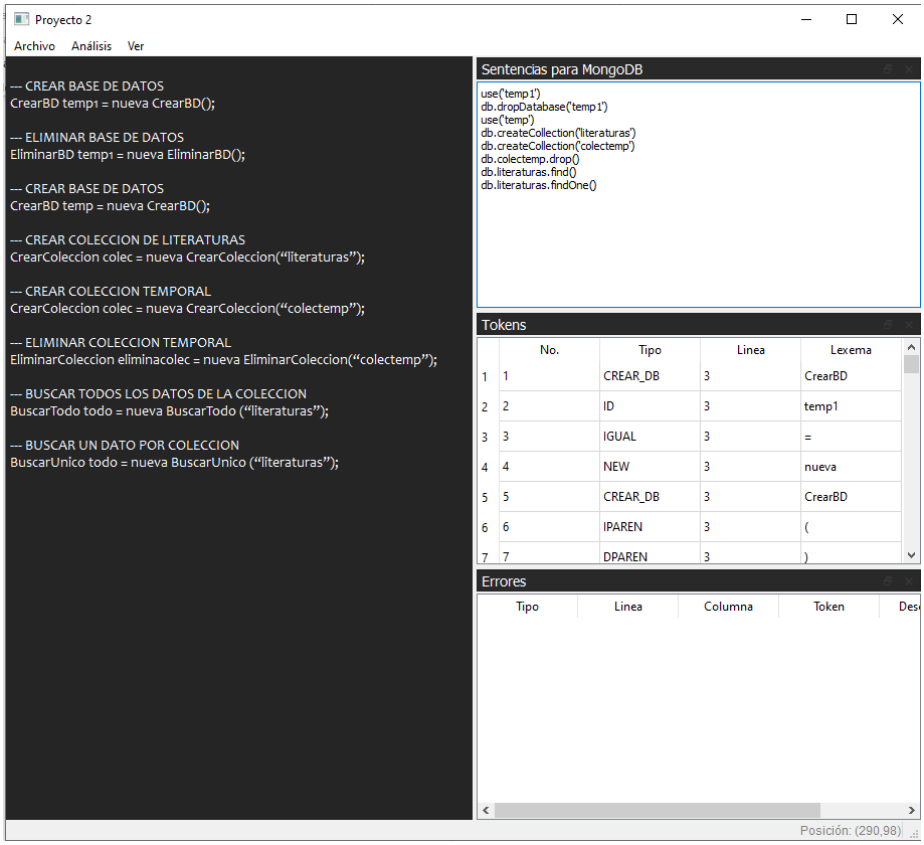
```
def analizar(self):
    input_str = self.cajaTexto.toPlainText()
    scanner_instance = scanner.Scanner(input_str)
    tokens = scanner_instance.initToken()  # Obtener los tokens
    parser = Parser(tokens)  # Crear una instancia del analizador con los tokens

    try:
        result = parser.parse()
        print(result)

        expresionesMongo = []
        for i in result:
            if i[0] == "CREAR_DB":
                expresionesMongo.append(f"use('{i[1]}')")
                print("CREAR_DB:", i[1])
            elif i[0] == "ELIMINAR_DB":
                expresionesMongo.append(f"db.dropDatabase('{i[1]}')")
                print("ELIMINAR_DB")
            elif i[0] == "CREAR_COLECCION":
                expresionesMongo.append(f"db.createCollection('{i[1]}')")
                print("CREAR_COLECCION:", i[1])
            elif i[0] == "ELIMINAR_COLECCION":
                expresionesMongo.append(f"db.{i[1]}.drop()")
            elif i[0] == "INSERTAR_UNICO":
                expresionesMongo.append(f"db.{i[1]}.insertOne({i[2]})")
            elif i[0] == "ACTUALIZAR_UNICO":
                expresionesMongo.append(f"db.{i[1]}.updateOne({i[2]}, {i[3]})")
            elif i[0] == "ELIMINAR_UNICO":
                expresionesMongo.append(f"db.{i[1]}.deleteOne({i[2]})")
```

Se define la función analizar, aquí se ejecuta el scanner (analizador léxico) donde comienza a leer la caja de texto y según la información contenida, se ingresa dentro de las expresiones Mongo generadas

Este botón ejecuta el AFD, lee el archivo de la caja de texto y comienza a analizar la estructura de datos, si dentro de la estructura se verifica que tiene caracteres erróneos, se almacenan en una tabla de errores, finaliza la ejecución de la lectura de datos y se imprimen los tokens en la tabla de tokens, y si existe una estructura especifica, se insertan las expresiones generadas en la tabla de Sentencias para MongoDB.



2.3.4. Ver -> Ver Tokens

```
def mostrarTokens(self, tokens):
    scanner = Scanner(self.cajaTexto.toPlainText())
    tokens = scanner.initToken()
    print("Tokens:", tokens)  # Imprime la variable tokens aquí para ver su valor

    if isinstance(tokens, list):
        self.tablaTokens.setRowCount(len(tokens))
        for i, token in enumerate(tokens):
            self.tablaTokens.setItem(i, 0, QTableWidgetItem(str(i + 1)))
            self.tablaTokens.setItem(i, 1, QTableWidgetItem(token[0]))
            self.tablaTokens.setItem(i, 2, QTableWidgetItem(str(token[2])))
            self.tablaTokens.setItem(i, 3, QTableWidgetItem(token[1]))
```

Se define la clase mostrarTokens, únicamente se ingresan los tokens que se encontraron dentro de la caja de texto principal al utilizar el scanner y se ingresan en la tabla de Tokens de la aplicación.

Visualiza únicamente los Tokens

Tokens

	No.	Tipo	Linea	Lexema
1	1	CREAR_DB	3	CrearBD
2	2	ID	3	temp1
3	3	IGUAL	3	=
4	4	NEW	3	nueva
5	5	CREAR_DB	3	CrearBD
6	6	IPAREN	3	(
7	7	DPAREN	3	)
8	8	PUNTOCOMA	3	;
9	9	ELIMINAR_DB	6	EliminarBD

2.3.5. Salir

Cierra la ventana de la aplicación.

2.3.6. Archivo Parser

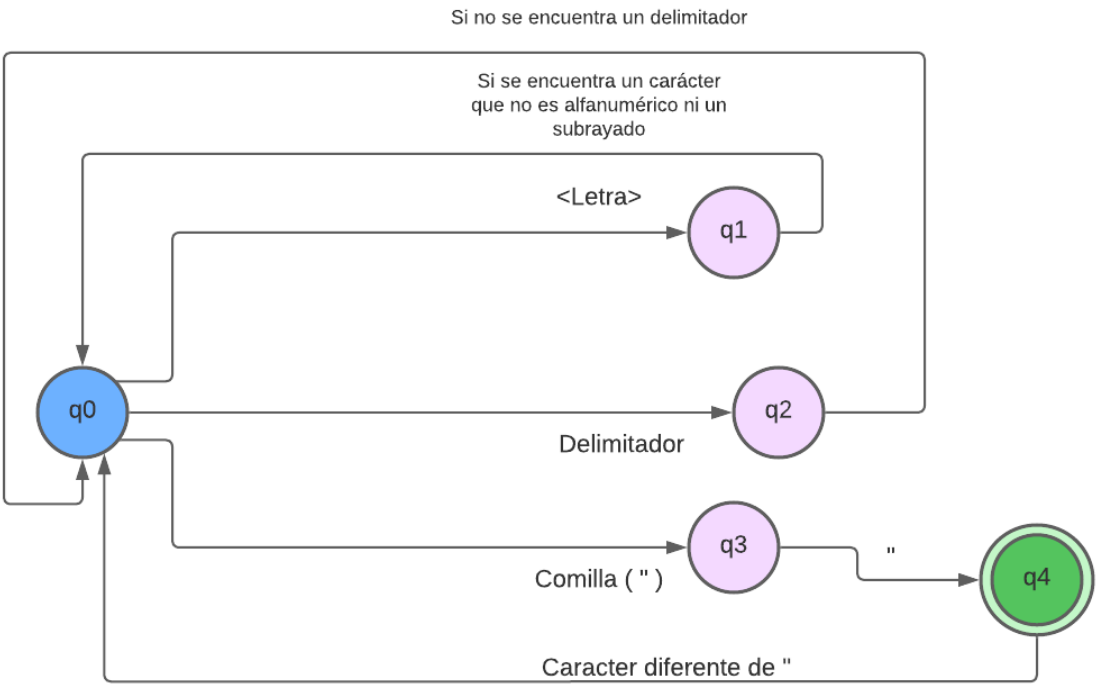
2.3.7. Archivo scanner

2.3.8. Tabla Tokens

Token	Descripción
CREATE_DB	Palabra clave para crear una base de datos
DROP_DB	Palabra clave para eliminar una base de datos
CREATE_COLLECTION	Palabra clave para crear una colección
DROP_COLLECTION	Palabra clave para eliminar una colección
INSERT_ONE	Palabra clave para insertar un documento en una colección
UPDATE_ONE	Palabra clave para actualizar un documento en una colección
DELETE_ONE	Palabra clave para eliminar un documento de una colección
FIND_ALL	Palabra clave para encontrar todos los documentos en una colección
FIND_ONE	Palabra clave para encontrar un documento en una colección
\s+	Uno o más espacios en blanco
[a-zA-Z0-9_]+	Una palabra que contenga solo letras (mayúsculas o minúsculas), números o guiones bajos
\s*	Cero o más espacios en blanco
(	Paréntesis de apertura
)	Paréntesis de cierre
“[a-zA-Z0-9_]+”	Una cadena entre comillas dobles que contenga solo letras (mayúsculas o minúsculas), números o guiones bajos
,\s*	Una coma seguida de cero o más espacios en blanco

2.3.9. Método del árbol

2.3.10. AFD



2.3.11. GLC

S -> CMD SPACES WORD SPACES ARGS

CMD -> CREATE_DB | DROP_DB | CREATE_COLLECTION | DROP_COLLECTION | INSERT_ONE | UPDATE_ONE | DELETE_ONE | FIND_ALL | FIND_ONE

SPACES -> ' ' SPACES | ' '

WORD -> LETTER WORD | LETTER

LETTER -> 'a' | 'b' | ... | 'z' | 'A' | 'B' | ... | 'Z' | '0' | '1' | ... | '9' | '_'

ARGS -> '(' SPACES STRING MORE_STRINGS SPACES ')' ARGS | ε

MORE_STRINGS -> ',' SPACES STRING MORE_STRINGS | ε

STRING -> "" CHARS ""

CHARS -> CHAR CHARS | CHAR

CHAR -> 'a' | 'b' | ... | 'z' | 'A' | 'B' | ... | 'Z' | '0' | '1' | ... | '9' | '_'