Universidad de San Carlos de Guatemala
Facultad de Ingeniería Escuela de Ingeniería en Ciencias y Sistemas
Organización de Lenguajes y Compiladores 2
Escuela de Vacaciones diciembre 2023
Catedrático: Ing. Luis Espino
Tutor académico: Ing. Juan Carlos Maeda

MANUAL TÉCNICO

Integrantes “Grupo 37”	Carné
Elder Ariel López Samol	201700404
Selvin Orlando Hernández	201700603
Luis Antonio Cutzal Chalí	201700841

Descripción general de la solución.

De acuerdo con el enunciado del proyecto, se creo una herramienta la cual tiene la capacidad para administrar bases de datos, capaz de manejar las instrucciones básicas de un DBMS relacional convencional, dicho sistema recibirá el nombre de XSQL. El servidor de base de datos cuenta con un IDE con el que el usuario interactúa directamente, cuenta con un conjunto de herramientas básicas que permiten el uso fácil de la herramienta.

Para almacenar y manipular la información de las bases de datos se maneja un sistema de archivos con formato XML, que contendrá la estructura de las bases de datos creadas, así como las tablas, funciones o procedimientos que contenga. Para la manipulación de la información se implementa el uso de un intérprete XML para leer la información guardada en los archivos.

Se utilizaron diversas librerías, métodos, clases y funciones para la creación optima del sistema.

Requerimientos mínimos del entorno del desarrollo.

1. Tener instalado Python versión 3.12.1 o la más actual.
2. Tener instalado cualquier editor de texto, de preferencia Visual Studio Code versión 1.85 o la más actual.
3. Tener instalado PLY ya que se utilizó dicha herramienta para escribir el analizador léxico y sintáctico utilizado en el proyecto.
4. Tener la librería de TKINTER para el IDE del proyecto.
5. Tener instalado Graphviz en su computadora.
6. Tener instalado el sistema operativo de Windows 10 o cualquier distribución de Linux o MAC.
7. Tener como mínimo 4 GB de memoria RAM en su computador.

Diccionario de clases, métodos y funciones:

En la herramienta se encuentran diversos tipos de clases, una de ellas es la parte del main el cual ejecuta la ventana inicial de la aplicación.

GUI_P.py: en esta clase se encuentran las funciones del entorno grafico de la aplicación.

- **tools_sql:** función para poder crear más querys en la ventana.
- **cerrar_pestana_actual:** cierra una query.
- **run_script:** ejecuta los comandos que se hayan escrito en el editor de texto.
- **update_gui:** esta función agrega nuevos datos al Treeview que se utiliza para mostrar en forma de árbol cada base de datos creada.
- **salir_programa:** termina la ejecución del programa.
- **guardar_como_archivo:** guarda lo que se haya escrito en cualquier pestaña del editor de texto
- **abrir_archivo:** carga en el área de las query cualquier archivo
- **mostrar_ventana_imagen:** está diseñada para mostrar una imagen llamada 'AST.png' utilizando el visor de imágenes del sistema operativo.
- **consola:** utiliza la biblioteca tkinter para crear una interfaz gráfica con un área de consola.

lexico.py: es una herramienta en Python que se utiliza para analizar y dividir un flujo de caracteres en tokens, que son unidades léxicas significativas. En términos sencillos, el analizador léxico PLY convierte una secuencia de caracteres en piezas más pequeñas y manejables llamadas "tokens".

sintactico.py: se utiliza para analizar la estructura gramatical de un programa, verificando si el código fuente sigue las reglas sintácticas del lenguaje de programación.

Arbol.py: relacionada con la representación y visualización de árboles de análisis sintáctico abstracto, recibe un nodo raíz y crea una instancia de la clase Arbol. También inicializa un entorno (enviroment) y lo guarda en la variable _env.

- **graficarAST:** genera un gráfico del árbol de análisis sintáctico abstracto (AST). Utiliza el método graficaNodos para construir una cadena que representa la estructura del árbol en formato DOT, que es utilizado por Graphviz para la visualización.
- **graficaNodos:** realiza un recorrido del árbol para construir la representación de nodos y relaciones entre ellos en formato DOT. Utiliza recursión para explorar los nodos y sus hijos.
- **generarGrafo:** toma la cadena DOT generada y la utiliza para crear un archivo que representa el árbol. Si el directorio de destino ya existe, simplemente se crea el archivo. Si no existe, se crea primero el directorio y luego se crea el archivo.

- **crearArchivo:** crea un archivo con la cadena DOT y luego ejecuta un comando de Graphviz para convertir el archivo DOT en una imagen gráfica del árbol.

AST.py: recibe una lista de instrucciones y crea una instancia de la clase AST. Estas instrucciones son típicamente nodos del árbol que representan la estructura sintáctica del código fuente,

- **getInstrucciones**: proporciona un medio para acceder a la lista de instrucciones almacenadas en la instancia del AST

Nodo.py: recibe parámetros que describen las propiedades del nodo, como el tipo de token, el lexema asociado, la línea y la columna en las que se encuentra en el código fuente. Además, inicializa una lista de hijos (_hijos) para almacenar nodos descendientes.

- **addHijo:** agrega un nodo a la lista de hijos del nodo actual. Esto permite la construcción gradual de la estructura del árbol, añadiendo nodos hijos a medida que se analiza el código fuente.
- **Lista de hijos:** almacena los nodos descendientes del nodo actual. Cada elemento de esta lista representa un subárbol o nodo hijo en la estructura jerárquica del árbol.

Fun_Concatena.py: proporciona una función específica, en este caso, para concatenar una lista de cadenas, El método devuelve la cadena resultante después de la concatenación.

- **getConcatena:** se inicia una variable _result como una cadena vacía. Luego, mediante un bucle, se recorre la lista de cadenas (a_concatenar), y se concatenan todas las cadenas para formar una sola cadena resultante.

Fun_Contar.py: proporciona una función específica para contar elementos que cumplen con una condición en un contexto de base de datos y tabla, El método devuelve el resultado como una cadena que representa la cantidad de elementos que cumplen con la condición (actualmente, siempre será cero porque la lista nDatos está vacía).

- **getContar:** toma tres parámetros: databaseName (nombre de la base de datos), tableName (nombre de la tabla) y condicion (condición para filtrar los elementos). Retorna el número de elementos que cumplen con la condición en la base de datos y tabla especificadas.

Dentro del método getContar, se inicializa una variable c en cero, que se utilizará para contar los elementos. Sin embargo, la lista nDatos que se crea está vacía, por lo que actualmente, no hay datos para contar.

Fun_Hoy.py: sirve para obtener la fecha y hora actuales y formatearlas como una cadena de texto, Se define un formato de fecha y hora en la variable formato como "%d-%m-%Y %H:%M:%S".

- **getToday:** se utiliza datetime.now() para obtener la fecha y hora actuales.

Fun_Substraer.py: sirve para obtener una subcadena de una cadena dada, especificando los índices de inicio y final.

- **getSubstraer:** toma tres parámetros: cadena (la cadena original), inicio (índice de inicio) y final (índice final). El método devuelve una subcadena que comienza en el índice inicio y termina en el índice final.

Antes de realizar la operación de obtención de subcadena, se realizan dos comprobaciones. Si el índice de inicio es menor que cero, se ajusta a cero. Si el índice final es mayor que la longitud de la cadena se ajusta a la longitud de la cadena.

Fun_Suma.py: clase diseñada para realizar la operación de suma en el contexto de una base de datos y una tabla específica.

- **getSuma:** toma tres parámetros: database (nombre de la base de datos), tablename (nombre de la tabla) y column (nombre de la columna sobre la cual se realizará la suma). El método devuelve la suma de los valores de la columna especificada en la tabla especificada de la base de datos.

Se inicializa la variable _result en 0.0. Esto sugiere que la suma se realizará en números decimales, el método devuelve el resultado de la suma, convertido a una cadena de texto.

Column.py: La clase column hereda de la clase Instruccion. El constructor (__init__) inicializa las propiedades de la columna, como el identificador (id), el tipo de dato (tipo), el tamaño (tamanio), la precisión (presicion), las restricciones (restricciones), y la posición en el código fuente (fila y colum), la clase almacena información sobre una columna, incluyendo su identificador, tipo de dato, tamaño, precisión y restricciones. Algunas propiedades tienen valores predeterminados si no se proporcionan (0 para tamanio y None para presicion).

ColumnWithoutFrom: inicializa una instancia de la clase ColumnWithoutFrom. Recibe tres parámetros: tableName (nombre de la tabla), columnName (nombre de la columna) y las posiciones en el código fuente (fila y columna), almacena información sobre una columna que no tiene una tabla especificada. Guarda el nombre de la tabla en la propiedad tableName y el nombre de la columna en la propiedad columnName.

CreateDB.py: hereda de la clase Instruccion. El constructor (__init__) inicializa las propiedades de la instrucción de creación de base de datos, como el nombre de la base de datos (dbName), la posición en el código fuente (fila y colum), y un manipulador XML (manipulador) que parece estar relacionado con operaciones de creación de base de datos.

- **Compilar:** esta destinado a realizar la compilación de la instrucción de creación de base de datos. En su implementación actual, agrega un nodo al árbol de análisis sintáctico (nodo) representando la instrucción de

creación de base de datos y, al mismo tiempo, utiliza el manipulador XML para llevar a cabo la creación de la base de datos.

La clase tiene una propiedad llamada manipulador que es una instancia de la clase CREATE_XML. Esto sugiere que la creación de la base de datos puede involucrar operaciones relacionadas con XML, pero los detalles específicos están en el método create_db del manipulador.

CreateSP.py: hereda de la clase Instruccion. El constructor (__init__) inicializa las propiedades de la instrucción de creación de procedimiento almacenado, incluyendo el nombre del procedimiento almacenado (sp_name), una lista de parámetros (parametros), una lista de instrucciones (instrucciones), y la posición en el código fuente (fila y colum).

- **Compilar:** esta destinado a realizar la compilación de la instrucción de creación del procedimiento almacenado. En su implementación actual, simplemente invoca al método compilar de la clase base (super().compilar(...)) y no realiza acciones adicionales.

CreateTable.py: hereda de la clase Instruccion. El constructor (__init__) inicializa las propiedades de la instrucción de creación de tabla, incluyendo el nombre de la tabla (identificador), el nombre de la base de datos aplicada (dbName), una lista de columnas (listaColumns), la posición en el código fuente (fila y colum), y un indicador de nueva línea (newLine).

- **Compilar:** es el responsable de compilar la instrucción de creación de tabla. En su implementación actual, crea un nodo en el árbol de análisis sintáctico (nodo) representando la instrucción y utiliza el manipulador XML para realizar operaciones de inserción en la base de datos.

Las columnas se representan mediante instancias de la clase column. El método compilar itera sobre las columnas, agrega nodos al árbol de análisis sintáctico y utiliza la información de las columnas para insertarlas en la base de datos.

Delete.py: hereda de la clase Instruccion. El constructor (__init__) inicializa las propiedades de la instrucción DELETE, como la posición en el código fuente (fila y colum).

- **Compilar:** es la implementación específica de la compilación para la instrucción DELETE.

Expresion.py: hereda de la clase Instruccion. El constructor (__init__) inicializa las propiedades de la expresión, como la posición en el código fuente (fila y colum).

- **Compilar:** es una implementación específica de compilación para expresiones.

Insert.py: hereda de la clase Instruccion. El constructor (__init__) inicializa las propiedades de la instrucción de inserción, como el nombre de la tabla

(tbl_name), las listas de columnas (lcolumns) y valores (lvalues) a insertar, la posición en el código fuente (fila y colum), y un manipulador XML (manipulador).

- **Compilar:** es el responsable de compilar la instrucción de inserción. En su implementación actual, crea nodos en el árbol de análisis sintáctico (nodo) representando la instrucción y utiliza el manipulador XML para realizar operaciones de inserción en la base de datos.

Antes de realizar la inserción, se verifica que la base de datos y la tabla especificadas existan utilizando el método exist_table del manipulador XML, se crean nodos en el árbol de análisis sintáctico para representar la información relacionada con la instrucción de inserción y ademas crea un objeto INSERT con la información relevante y se utiliza el manipulador XML para realizar la operación de inserción en la base de datos.

Reference: hereda de la clase Instruccion. El constructor (__init__) inicializa las propiedades de la referencia, como el nombre de la tabla de referencia (tablaReference), el nombre de la columna de referencia (columReference), y la posición en el código fuente (fila y colum).

- **Compilar:** es una implementación específica de compilación para la referencia. En su implementación actual, simplemente invoca al método compilar de la clase base (super().compilar(tree, tablaSim, nodo, output)).

La llamada a super().compilar(tree, tablaSim, nodo, output) indica que se está invocando al método compilar de la clase base (Instruccion).

Restriccion.py: hereda de la clase Instruccion. El constructor (__init__) inicializa las propiedades de la restricción, como el tipo de restricción (tipoRest), y la posición en el código fuente (fila y colum).

Select.py: hereda de la clase Instruccion. El constructor (__init__) inicializa las propiedades de la instrucción de selección, como si involucra varias tablas (sonVariasTablas), la lista de tablas (listaTablas), la lista de columnas (columnas), la base de datos aplicada (dbAplied), la lista de condiciones (condiciones), y la posición en el código fuente (fila y colum).

- **Compilar:** crea nodos en el árbol de análisis sintáctico (nodo) representando la instrucción y utiliza el manipulador XML para realizar operaciones de selección en la base de datos.

Se crean nodos en el árbol de análisis sintáctico para representar la información relacionada con la instrucción de selección, incluyendo la lista de columnas, la lista de tablas y la lista de condiciones.

Update.py: hereda de la clase Instruccion. El constructor (__init__) inicializa las propiedades de la instrucción de actualización, como la posición en el código fuente (fila y colum).

- **Compilar:** es una implementación específica de compilación para la instrucción de actualización. En su implementación actual, simplemente invoca al método compilar de la clase base (super().compilar(tree, table)).

Use.py: hereda de la clase Instruccion. El constructor (__init__) inicializa las propiedades de la instrucción USE, como el nombre de la base de datos (db_name), y la posición en el código fuente (fila y colum).

- **Compilar:** crea nodos en el árbol de análisis sintáctico (nodo) representando la instrucción y utiliza el manipulador XML para verificar la existencia de la base de datos y realizar operaciones de uso.

Se crea un nodo en el árbol de análisis sintáctico para representar la instrucción USE. El nodo contiene información sobre el nombre de la base de datos y verifica la existencia de la base de datos utilizando el manipulador XML. Si la base de datos existe, se establece el valor de la variable useDB en el ámbito, indicando que se ha seleccionado una base de datos.

Variable.py: hereda de la clase Instruccion. El constructor (__init__) inicializa las propiedades de la variable, como el identificador (id), el tipo de variable (tipo), y la posición en el código fuente (fila y colum).

- **Compilar:** invoca al método compilar de la clase base (super().compilar(tree, tablaSim, nodo, output)).

La llamada a super().compilar(tree, tablaSim, nodo, output) indica que se está invocando al método compilar de la clase base (Instruccion).

Ámbito.py: El constructor (__init__) inicializa un ámbito con un posible ámbito padre (ambitoPadre). Si no se proporciona un ámbito padre, se establece en None. Además, inicializa una lista de símbolos (simbolos) vacía para almacenar los símbolos asociados con el ámbito.

- **addSimbolo:** agrega un símbolo al ámbito.
- **getValueFromSimbolo:** busca y devuelve el símbolo correspondiente al identificador proporcionado (id).
- **setValueFromId:** busca un símbolo por su identificador y actualiza su valor con el nuevo valor proporcionado (newValue).

manipulador_xml.py: Inicializa atributos como _db_name, _tb_name, root, y _colums en None o una lista vacía.

- **create_db:** Crea una nueva base de datos en el archivo XML o agrega una nueva si el archivo ya existe, si el archivo XML no existe, crea un nuevo elemento raíz y agrega la base de datos.
- **insert_db:** Inserta una nueva base de datos en el archivo XML existente y utiliza el método cuando el archivo XML ya existe.
- **insert_table:** inserta una nueva tabla en la base de datos existente en el archivo XML y agrega información sobre las columnas de la tabla.
- **exist_table:** Verifican la existencia de una tabla en el archivo XML.
- **exist_db:** Verifican la existencia de una base de datos en el archivo XML.

- **Insert_ontbl:** Realiza la operación de inserción de datos en una tabla existente además de realizar la inserción de datos en las columnas correspondientes.
- **Compare_type:** Compara el tipo de datos de una columna con el valor proporcionado para la inserción.
- **select:** Realizan operaciones de selección de datos.
- **select_all:** muestra todos los datos de las columnas en una tabla específica.
- **xml_gui:** Toma un nombre de archivo XML como entrada y devuelve información sobre las bases de datos y tablas en formato de lista.
- **delete_db:** Elimina una base de datos del archivo XML.
- **mensaje_error:** Muestra un mensaje de error utilizando la interfaz gráfica de Tkinter.
- **createDump:** Crea un archivo XML de respaldo (_DUMP.xml) con la misma estructura que la base de datos original.
- **compy_element:** Copia elementos XML de un elemento original a uno nuevo.
- **insert_db2:** El método copia las tablas, funciones y procedimientos de la base de datos original al nuevo archivo de respaldo. Si ya existe un respaldo con el mismo nombre, se muestra un mensaje de error.

table.py: inicializa la tabla con un nombre de base de datos (db_name) y un nombre de tabla (tbl_name).

- **inisert_column:** Agrega una columna a la lista de columnas de la tabla el cual recibe como parámetro un objeto column que representa una columna de la tabla.
- **get_db_name:** proporcionan acceso a los nombres de la base de datos.
- **get_tb_name:** proporcionan acceso a los nombres de la tabla.