Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Lenguajes Formales y de Programación
Segundo Semestre, 2020

SimpleQL CLI

Objetivos

- ### General

  Aplicar conceptos de lenguajes formales y de programación, sus paradigmas y AFDs en la implementación de una solución de software.

- ### Específicos

  1. Utilizar Python como lenguaje de programación para el desarrollo de la solución.
  2. Implementar una interfaz de línea de comandos cuyas instrucciones y comandos se basen en el paradigma declarativo de los lenguajes de programación.
  3. Utilizar AFDs para interactuar tanto con SimpleQL como con los archivos de texto que contienen la información a cargar.

Descripción

SimpleQL CLI es una interfaz de línea de comandos que permite utilizar SimpleQL para realizar diferentes operaciones de análisis y consultas sobre un conjuto de datos que se encuentra alojado en memoria. Puede verse como una versión 2.0 del SimpleQL original, con la diferencia de que esta versión amplía las capacidades de análisis, consulta y búsqueda, al mismo tiempo que permite trabajar con registros que tengan estructuras diferentes. Esta versión de SimpleQL también implementa su propia notación de objetos para los archivos de texto que contienen la información inicial, esta se conoce como AON (Alternative Object Notation) y su estructura se define más adelante.

Otra de las particularidades con las que cuenta SimpleQL CLI es que pretende minimizar los errores que pudieran ser generados al un usuario introducir caracteres extraños, por lo que tiene la capacidad de ignorar espacios y caracteres ajenos a sus instrucciones y comandos.

SimpleQL CLI sigue siendo case insensitive y provee ahora la capacidad de ejecutar scripts con consultas para no tener que escribir estas una por una en la línea de comandos.

Funciones y Comandos

Notación e instrucciones preliminares:

< >	Estos caracteres indican que su contenido es determinado por el usuario, por ejemplo, < ID > podría tomar como valores válidos: Estudiantes, Mascotas, Clientes, etc.
+	Indica que el elemento puede venir de una a muchas veces.
Palabras en mayúscula	Indican keywords que pertenecen a SimpleQL
[ ]	Indican que su contenido es opcional

1. **CREATE SET < ID >**

   Tiene la función de crear sets de memoria donde se alojarán ciertos conjuntos de datos cargados por el usuario. La aplicación tiene la capacidad de poseer activos N conjuntos de datos. Ejemplos:

   CREATE SET carros

   CREATE SET elementos

2. **LOAD INTO < set_id > FILES < id > [, <id>] +**

   Este comando carga al conjunto especificado por set_id la información contenida en la los archivos de la lista de archivos definida después de la keyword FILES. Ejemplos:

   LOAD INTO elementos FILES periodica.aon, periodica2.aon

   LOAD INTO carros FILES carros.aon

   Algunas consideraciones para tomar en cuenta son que los archivos para cargarse a un mismo set siempre tendrán la misma estructura, es decir, en ningún momento se tendrá que cargar al set carros un archivo con información de elementos y otro archivo con información de carros, garantizando así que todos los registros de un set tengan siempre la misma estructura.

3. **USE SET < set_id >**

   Este comando define el set de datos a utilizar para las siguientes operaciones, si se intenta realizar operaciones sin haber definido un set de datos la aplicación debe mostrar un error. Ejemplo:

   USE SET carros

   USET SET elementos

4. **SELECT < atributo > [, <atributo>] + [WHERE < condiciones >]**

   Permite seleccionar uno o más registros o atributos de los mismos con base en condiciones simples que pueden aplicarse a los atributos de los mismos.

   En lugar de listar los atributos también es posible utilizar el operador *, esto automáticamente seleccionará todos los campos del registro.

   Ya que la estructura de los sets no está predefinida, sino que viene definida en los archivos los atributos seleccionables serán cualquiera que pertenezca al set sobre el cual se está actualmente trabajando.

   Algunos Ejemplos para el set carros:

   SELECT modelo, tipo, marca, año WHERE color = “rojo”

   SELECCIONAR *

   SELECCIONAR * WHERE marca = “Mazda” AND año < 1996

   Otras funcionalidades con las que cuenta el comando SELECT es la ampliación de las condiciones, a continuación se definen las ampliaciones:

   - Es posible utilizar diferentes operaciones de comparación, las mismas que serían utilizadas en un lenguaje regular de programación: < (menor que), > (mayor que), <= (menor igual), >= (mayor igual), = (igual) y != (no igual). Estas operaciones solo pueden ser realizadas entre datos del mismo tipo: cadenas con cadenas, números con números y booleanos con booleanos (en el caso de los booleanos True es siempre mayor que False). En caso de comparar cadenas se hará de forma lexicográfica.
   - Es posible combinar condiciones utilizado los operadores AND (conjunción), OR (disyunción) y XOR (disyunción exclusiva).
   - El comando SELECT permitirá el uso de expresiones regulares, las reglas de las mismas se definen más adelante.

   Algunos adicionales para el set de elementos:

   SELECT nombre, padre, siglas WHERE tipo = “metal” OR tipo = “carbono”

   SELECCIONAR * WHERE siglas = “HG”

5. LIST ATTRIBUTES

Este comando permite listar los atributos que componen a cada registro del set.

6. PRINT IN <color>

Este comando permite al usuario elegir el color en el que serán presentados los resultados en la línea de comandos. Los valores a elegir serán BLUE, RED, GREEN, YELLOW, ORANGE y PINK. Ejemplo:

PRINT IN BLUE

7. MAX < atributo > | MIN < atributo >

Permiten encontrar el valor máximo o el valor mínimo que se encuentre en el atributo de uno de los registros del conjunto en memoria. En caso de seleccionar el valor máximo de un valor de tipo String la comparación será realizada de forma lexicográfica. Ejemplos usando el set carro:

MAX año

MIN modelo

8. SUM < atributo > [, <atributo>] +

Permite obtener la suma de todos los valores de un atributo especificado en el comando. Este comando solamente se utilizará sobre valores de tipo numérico, no se realizarán sumas sobre valores de tipo cadena o booleanos. En caso de seleccionarse varios atributos deberá reportar cada atributo con su respectiva suma, en caso de que el atributo tenga valor null se ignorará. El comando SUM acepta el uso del operador *.

Ejemplos:

SUM edad, promedio, faltas

SUM asistencias

9. **COUNT < atributo > [, < atributo >] +**

   Permite contar el número de registros que se han cargado a memoria. En caso de que alguno de los atributos tenga valor null se ignorará. El comando COUNT permite el uso del operador *.

   Ejemplos:

   COUNT *

   COUNT edad, promedio, faltas

10. **REPORT TO < id > < comando >**

    Este comando permite crear un reporte en html a partir de cualquier otro comando de análisis o selección. El reporte debe ser agradable a la vista y fácil de leer. El id define el nombre del archivo sobre el que se crea el reporte.

    Ejemplos:

    REPORT TO reporte1 COUNT *

    REPORT TO reporte2 SUM *

    REPORT TO reporte3 SELECT * WHERE edad != 44

11. **SCRIPT < direccion > [, < direccion >]**

    Este comando permite cargar scripts con extensión .siql que contienen series de instrucciones y comandos SimpleQL, esto con el objetivo de que el usuario no tenga que escribir uno por uno los comandos que se desee ejecutar. Algunos lineamientos para este tipo de archivo se dan más adelante.

12. **REPORT TOKENS**

    Este comando crea un reporte en html que muestra una lista de todos los lexemas encontrados por el AFD, mostrando también a cual token pertenece el lexema y una breve descripción del mismo.

Consideraciones de la CLI

1. Es de vital importancia que los resultados sean presentados de forma entendible y ordenada, pues es el objetivo de la aplicación facilitar al usuario la búsqueda y manipulación de información.
2. Se recomienda utilizar algún caracter como separador de resultados, tal como el guión (-), el numeral (#) o el signo igual (=)

Alternative Object Notation (AON)

AON o Alternative Object Notation es una notación de objetos exclusiva de SimpleQl CLI, la misma cuenta con una estructura básica en la que son permitidos arreglos, objetos y atributos de los mismos, en cierta forma es muy similar a JSON, pero con ciertas características diferentes, a continuación se definen las reglas básicas de AON:

( )	Los paréntesis definen un arreglo, todo lo que se encuentra adentro es un elemento del mismo
< >	Estos símbolos definen un objeto, los atributos de un objeto están separados por comas.
[ ]	Se utilizan para definir identificadores, o en otras palabra el nombre de un atributo.

Para dar una idea más clara de AON se provee un ejemplo del mismo disponible en el siguiente link.

*Nota: Se debe hacer uso de un AFD para poder obtener y cargar la información del archivo.

SimpleQL Regex

Una de las particularidades de SimpleQL es que permite utilizar regex para encontrar coincidencias sobre cadenas para búsquedas en la clausula WHERE. Las reglas se definen a continuación:

+	Define que el elemento anterior puede venir entre una y muchas veces.
*	Define que el elemento anterior puede venir desde cero a muchas veces.
?	Define que el elemento anterior puede venir una vez o no venir.
( )	Se utilizan para agrupar elementos, al estar agrupados se toman como uno solo al aplicar cualquier otro de los operadores.
|	Se utiliza para separar opciones, es decir que en una cadena pueda venir el elemento a la izquierda o el elemento a la derecha.
^	Se utiliza para definir cuál será el primer elemento en la cadena
[ ]	Son los caracteres que rodean a la regex para delimitarla
.	Denota cualquier caracter

Las regex funcionan únicamente buscando coincidencias, es decir, cuando se use se deberá devolver todos los registros cuya cadena comparada concuerde con la regex.

*Nota: Las regex SI son case sensitive, A y a no son lo mismo.

**Nota: Las regex no toman en cuenta espacios en blanco, saltos de linea u otros caracteres invisibles.

Ejemplos:

SELECT * WHERE nombre REGEX [^a|^b|^O|ab.+]

El ejemplo anterior concuerda con todos los nombres que empiecen con a, con b, con O o que empiecen con a seguido de b y cualquier otro caracter por lo menos una vez.

SimpleQL Script (Siql)

Se trata de archivos de script que contienen instrucciones previamente escritas para que SimpleQL pueda ejecutarlas. Para facilitar la separación de los comandos, al final de cada uno vendrá un punto y coma como es usual en lenguajes de programación.

Entregables

- Manual de Usuario en formato .md, este debe ser escrito en el archivo README del repositorio. El mismo debe enseñar a una persona a utilizar la aplicación desde cero describiendo como utilizar los comandos, las regex y la notación AON.
- Código Fuente
- Manual Técnico, con una explicación clara del flujo de su aplicación, este debe ser entregado directamente en la plataforma de UEDi en formato PDF.

Consideraciones

- Totas las dudas deben ser realizadas por medio de los canales oficiales de comunicación: Uedi, Slack o j.alberto.cabrerapuente@gmail.com.
- La aplicación debe ser realizada utilizando Python como lenguaje de programación.
- El reporte en html debe ser entendible y atractivo a la vista.
- La práctica debe ser desarrollada en forma individual.
- La entrega se realizará por medio de un repositorio localizado en Github, el repositorio debe ser privado y debe ser compartido con el usuario **cabreration**,, el link al mismo se entregará en Uedi.
- No se permite la modificación de código durante la calificación.
- No se permite la modificación de los archivos de entrada durante la calificación
- Cualquier modificación a comandos durante la calificación tendrá una penalización en los puntos.
- Copias totales o parciales tendrán una nota automática de 0 puntos y serán notificadas a la escuela de Ciencias y Sistemas.
- **La entrega del proyecto será el 04 de Octubre, a más tardar a las 23:59.**