TP Bases de Datos II

Introducción

El presente trabajo consistirá en desarrollar iterativamente una base de datos relacional sumamente simple y de características muy limitadas, con el objetivo de entender el funcionamiento interno de las mismas, conocer las estructuras de datos involucradas y las problemáticas que motivaron el diseño de las mismas.

En cada iteración presentaremos una extensión del enunciado que sumará requerimientos al producto que venimos construyendo, o extenderá los existentes.

El lenguaje de programación a utilizar quedará a elección de cada alumno entre las opciones que den los docentes. Cada proyecto deberá respetar en cada entrega las condiciones enunciadas. Esto es fundamental para la evaluación del mismo.

Se requerirá que todos los proyectos pasen una serie de tests automatizados.
Estos tests verificarán el output del programa por lo cual el mismo deberá respetar estrictamente las especificaciones establecidas por el enunciado.

En la última entrega se deberá defenderá el trabajo

Parte 1

(ENTREGA 1)
En esta primera parte del trabajo práctico nos enfocaremos en el setup del entorno y la implementación de un par de utilidades básicas.
Límite de tiempo: 1 semana

Requerimientos para la entrega

1. Elección de lenguaje de implementación
2. Inicialización del repositorio de código
3. Implementación del REPL
4. Compilador y máquina virtual

1- Elección de lenguaje de implementación

Las entregas podrán ser realizarse en alguno de los siguientes lenguajes:

- C/C++
- Python

La elección del lenguaje en el que van a implementar el proyecto es importante porque las características del mismo pueden simplificar o dificultar ciertas tareas. Es importante elegir un lenguaje en el que se sientan cómodos, pero no olvidar que tendremos que manipular la

memoria de manera precisa. Algunos lenguajes de más alto nivel pueden ser más cómodos que otros para describir soluciones rápidamente, pero puede volverse más trabajoso manipular tipos de datos específicos.

2- Inicialización del repositorio de código

Para la entrega deberán crear un repositorio en GITHUB y darnos acceso (o hacerlo público).
El mismo deberá contener:

- Un *Makefile* (actualizado en cada entrega) que defina la regla "*deps*" , que se encargue de la instalación de dependencias y cualquier otra tarea necesaria
- Un shell script (un script ejecutable en bash) de nombre *run.sh* que ejecute el programa que implementaremos

3- Implementación del REPL

El objetivo final de este trabajo es desarrollar una base de datos similar a SQLite, así que empezaremos por el REPL (Read-Execute-Print Loop) que nos permitirá ejecutar queries y comandos, así como evaluar cada entrega. Cómo su nombre lo indica, el REPL es un programa que:

1. acepta un input de entrada (READ)
2. ejecuta esa entrada (EXECUTE)
3. imprime el resultado (PRINT)
4. vuelve a comenzar (LOOP)

En definitiva sería como la aplicación de consola sqlite3, que permite ejecutar sentencias sql o comandos .

Como usuarios queremos que al ejecutar este programa, imprima al standard output (*stdout*) el siguiente prompt

```
sql>
```

No deberán imprimirse espacios ni otros caracteres antes ni después del prompt. Es importante seguir estas especificaciones al pie de la letra ya que los tests automatizados que usaremos para evaluar el funcionamiento de las entregas dependen fuertemente del formato del output del programa.

También queremos que al ingresar un comando ante el prompt, el programa responda imprimiendo al stdout el resultado de la operación, y luego presente nuevamente el prompt en una nueva línea. Por ej:

```
sql>.comando
resultado
sql>
```

De la misma manera, no deberán imprimirse espacios u otros caracteres al comienzo o fin de cada línea del resultado.

4- Compilador y Máquina Virtual

El compilador de SQL es un programa que parsea un string y devuelve una representación interna de la sentencia parseada, llamado bytecode.
La máquina virtual toma ese bytecode y ejecuta la query o el comando que este representa.
El compilador que desarrollaremos a lo largo de las entrega sabrá interpretar dos tipos diferentes de sentencias:

- **sentencias SQL** (trivial)
- **meta-comandos**: comandos no sql que brindarán información sobre el estado la base de datos en sí misma, o que realizarán alguna acción sobre el programa en ejecución, por ejemplo, en SQLite, **.exit** termina el programa, **.tables** lista las tablas existentes en la base de datos actual, etc. Como se puede ver en los ejemplos mencionados, los *meta-comandos* se distinguen prefijando un punto a la keyword del comando, para hacer más fácil la distinción entre estos y las sentencias SQL.

Requerimientos

- Metacomandos
  - `.exit`: El frontend deberá entender este comando y como efecto debe imprimir al stdout el string "Terminado" , terminar el programa en ejecución, volviendo así al shell del sistema.
    Por ej:
    ```
    sql>.exit
    Terminado
    $
    ```
- Sentencias SQL
  - `select`: El frontend deberá aceptar este comando y como efecto del mismo debe imprimir al stdout el string "SELECT no implementadoo"
    Por ej:
    ```
    sql>select
    SELECT no implementado
    $
    ```
  - `insert`: El frontend deberá aceptar este comando y como efecto del mismo debe imprimir al stdout el string "INSERT no implementado"
    Por ej:
    ```
    sql>insert
    INSERT no implementado
    ```