Estudiantes:
Sebastian Bolaños Zamora (2024099520)
Gabriel David Soto López (2024178797)

Docente: Jose Isaac Ramirez Herrera

Curso: Algoritmos Y Estructuras De Datos I

Carrera: Ingeniería en computadores

Universidad: Instituto Tecnologico Costa Rica

PROYECTO II

TinySQLDb

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computadores
Algoritmos y Estructuras de Datos I (CE 1103)
II Semestre 2024

TEC | Tecnológico de Costa Rica

Introducción

Este documento describe la solución implementada para la interacción entre un cliente en PowerShell y un servidor que administra un sistema de base de datos simulado mediante archivos en el sistema de archivos. A través de diversas sentencias SQL, el sistema permite crear bases de datos, gestionar tablas, ejecutar consultas y realizar operaciones sobre los datos, todo a través de un cliente en PowerShell 7. Se abordarán los requerimientos de la solución, los desafíos y las decisiones de diseño adoptadas.

Tabla de contenidos

Breve descripción del problema

El problema a resolver consiste en la creación de un sistema que simule las funcionalidades básicas de un gestor de bases de datos, tales como la creación de bases de datos, la creación de tablas dentro de estas bases de datos, la selección de una base de datos para realizar operaciones, y la eliminación de tablas. El sistema debe ser capaz de manejar estos elementos de forma persistente, utilizando archivos en el sistema de archivos del equipo para almacenar la información.

La simulación de este sistema se realiza sin utilizar bases de datos comerciales ni librerías externas para bases de datos, lo que implica la manipulación directa de archivos binarios en el sistema de archivos.

Descripción de la solución

Cliente en PowerShell

- El cliente es un módulo en PowerShell que permite ejecutar comandos SQL a través de la función Execute-MyQuery. Esta función acepta tres parámetros:
    - QueryFile: un archivo con sentencias SQL.
    - Port: el puerto donde el servidor está escuchando.
    - IP: la dirección IP del servidor.

  Cada sentencia SQL en el archivo se ejecuta una por una, mostrando los resultados en formato de tabla en la terminal. Además, el tiempo de ejecución del servidor para cada sentencia se reporta. Las sentencias están separadas por punto y coma (;).

- **Alternativa considerada:** Se consideró usar otros lenguajes para el cliente, como Python, pero se optó por PowerShell debido a su capacidad de formatear

resultados en tablas, su compatibilidad con sistemas Windows, y su facilidad de uso para la automatización de tareas.

- **Limitación:** La implementación actual solo permite la ejecución de sentencias predefinidas dentro de un archivo y no acepta comandos en tiempo real ingresados directamente en la consola.
- **Problema:** No se ha encontrado ninguno relevante.

Sistema de base de datos basado en archivos

- **Implementación**: Se decidió usar carpetas para representar bases de datos y archivos binarios para tablas y metadata. El system catalog usa archivos binarios para almacenar información de bases de datos, tablas y columnas.
- **Alternativas**: No hay otra alternativa para hacer la base de datos.
- **Limitaciones**: Menor eficiencia al gestionar grandes cantidades de datos comparado con sistemas de bases de datos tradicionales.
- **Problemas**: La sincronización de los archivos al manipular la metadata podría volverse lenta.

CREATE DATABASE

- **Implementación**: Para crear una base de datos, el cliente ejecuta la sentencia CREATE DATABASE <database-name>. Se extrae el nombre de la base de datos usando extractor.ExtractDatabaseName(sentence) y se crea una nueva instancia de la clase CreateDatabase, que realiza la operación.
- **Alternativas**: No hay otra alternativa para hacer el CREATE DATABASE.
- **Limitaciones**: El nombre de la base de datos no puede contener caracteres especiales, lo que limita la flexibilidad en los nombres.
- **Problemas**: Controlar errores de permisos en el sistema de archivos.

SET DATABASE

- **Implementación**: La sentencia SET DATABASE <database-name> establece el contexto para futuras operaciones en una base de datos específica. El cliente envía esta instrucción al servidor, que valida si la base de datos existe. El contexto del cliente se actualiza en función de la respuesta del servidor.
- **Alternativas**: Uso de variables globales o conexiones persistentes a bases de datos.
- **Limitaciones**: Depende de que el sistema de archivos mantenga las carpetas consistentes.
- **Problemas**: Errores al cambiar de base de datos si está corrompida.

CREATE TABLE

- **Implementación**: Para crear una tabla, el cliente ejecuta CREATE TABLE <table-name> AS (column-definition). El cliente extrae los parámetros usando extractor.ExtractCreateTableParameters(sentence) y luego llama a la clase CreateTable para ejecutar la operación.
- **Alternativas**: No hay otra alternativa para hacer el CREATE TABLE.
- **Limitaciones**: Las tablas se limitan a los tipos de datos definidos (INTEGER, DOUBLE, VARCHAR, DATETIME).
- **Problemas**: El manejo de restricciones de tipo de dato y la longitud de las columnas fue un desafío técnico importante.

DROP TABLE

- **Implementación**: El cliente puede eliminar una tabla con DROP TABLE <table-name>, siempre que la tabla esté vacía. Se extrae el nombre de la tabla con extractor.ExtractTableName(sentence) y se ejecuta la operación.
- **Alternativas**: Permitir eliminación de tablas no vacías.

- **Limitaciones**: Requiere verificar si la tabla tiene datos antes de eliminarla.
- **Problemas**: Manejar errores al intentar eliminar una tabla con datos.

INSERT INTO

- **Implementación**: La sentencia INSERT INTO <table-name> VALUES (<values>,<values>...) permite insertar filas en una tabla. Se extraen los valores de inserción usando extractor.ExtractInsertParameters(sentence) y se ejecuta la operación con una instancia de Insert.
- **Alternativas**: Insertar sin validar tipos de datos, delegando esta tarea al cliente.
- **Limitaciones**: La inserción es secuencial, lo que puede ser lento en tablas grandes.
- **Problemas**: Manejo de errores al insertar datos con tipos incorrectos o duplicados en columnas indexadas.

Operación SELECT

- **Implementación**: La operación SELECT se implementó para leer archivos binarios que representan las tablas. Se utiliza un FileStream junto con BinaryReader para extraer los datos y luego imprimirlos. En el caso de SELECT *, el sistema lee todas las columnas de cada fila.
- **Alternativas:** Se evaluó usar bases de datos existentes como SQLite o almacenar los datos en archivos de texto en lugar de binarios, pero se optó por un enfoque personalizado para mantener el control sobre los procesos.
- **Limitaciones**: Actualmente el comando solo lee todos los datos presentes en la tabla.
- **Problemas**: Se encontraron problemas al intentar leer más allá del archivo. También hubo problemas con la lectura incorrecta de enteros.

Operaciones "UPDATE", "DELETE" e "INDEX":

Estas operaciones están implementadas parcialmente y, por ahora, lanzan excepciones NotImplementedException. Sin embargo, se intentó aplicar la validación de datos y la extracción de parámetros mediante expresiones regulares para asegurar que las sentencias proporcionadas cumplieran con la estructura esperada.

Validación de Datos

- En cada operación, se validan los tipos de datos definidos para las columnas con los valores proporcionados. Esto evita que se inserten datos inconsistentes en las tablas.

Extracción con Expresiones Regulares

- Se utilizaron expresiones regulares para identificar los componentes clave de las sentencias SQL (como el nombre de la tabla, columnas y valores). Aunque esto fue efectivo para sentencias simples, se reconoció que una solución más compleja sería necesaria para manejar consultas más avanzadas.

Repositorio del proyecto:

https://github.com/GabrielSL24/ProjectTinySQLD.git

Diseño general

Diagrama de clases UML