Instituto Tecnológico de Costa Rica

Ingeniería en Computadores

Algoritmos y Estructuras de Datos I

Proyecto II : TinySQL

Grupo : 1

Profesor :

Leonardo Araya

Miembros de Grupo :

Jiacheng Tan He
Justin Solano Calderon

Octubre 7, 2024

Tabla de Contenidos

Introducción

Este proyecto tiene como objetivo principal el diseño y la implementación de un motor de bases de datos relacional sencillo, denominado TinySQLDb, con el propósito de familiarizar a los estudiantes con el funcionamiento básico de estos sistemas. Se utiliza como plataforma de enseñanza en el curso de Algoritmos y Estructuras de Datos I, del Instituto Tecnológico de Costa Rica, y busca integrar los conocimientos adquiridos sobre estructuras de datos, algoritmos y programación orientada a objetos en un contexto práctico.

Descripción del problema

El problema central radica en la implementación de un sistema administrador de bases de datos que permita la ejecución de consultas SQL a través de una interfaz cliente-servidor. El cliente debe ser capaz de enviar consultas al servidor, que procesará y ejecutará las instrucciones utilizando un conjunto de componentes fundamentales: el procesador de consultas, el manejador de datos almacenados y la interfaz de comunicación. Adicionalmente, el sistema debe gestionar eficientemente las operaciones en disco, optimizando la recuperación de datos mediante la creación de índices. El desafío técnico se centra en diseñar una solución que permita gestionar las bases de datos, las tablas y las columnas asociadas, todo esto manteniendo un enfoque orientado a objetos en el lenguaje de programación C#.

Descripción de la solución

Punto 000

El cliente se implementa como un módulo en PowerShell 7, el cual permite la ejecución de comandos SQL sobre una interfaz API desde la terminal de PowerShell. La función principal, **Execute-MyQuery**, recibe tres parámetros clave: **QueryFile**, que es la ruta al archivo de script SQL con las sentencias a ejecutar, **Port**, que indica el puerto en el cual el API escucha, y **IP**, que representa la dirección IP del servidor de la interfaz API. La función lee el archivo SQL y separa las sentencias individuales utilizando el punto y coma (;) como delimitador. Luego, para cada sentencia SQL, se realiza una conexión al servidor especificado utilizando el puerto y la IP proporcionados, enviando cada comando de forma secuencial. El tiempo de ejecución de cada sentencia se mide y se muestra en la terminal, junto con el resultado, que es formateado como una tabla gracias a las funciones de PowerShell para mostrar objetos en ese formato. Finalmente, los resultados de cada sentencia son mostrados uno por uno, proporcionando un formato de salida fácil de leer para el usuario.

Punto 001

En el código realizado, una base de datos se representa como una carpeta en el sistema de archivos, donde cada tabla dentro de la base de datos se guarda como un archivo txt en esa carpeta. El catálogo del sistema, es una carpeta compartida por todas las bases de datos y contiene varios archivos que almacenan la metadata necesaria para gestionar las bases de datos. Estos archivos son:

1. **SystemDatabases.txt**: contiene el listado de las bases de datos existentes.
2. **SystemTables.txt**: guarda la información de las tablas existentes en cada base de datos.
3. **SystemColumns.txt**: almacena información sobre las columnas de cada tabla.
4. **SystemIndexes.txt**: contiene información sobre los índices asociados a cada tabla.

Este catálogo permite realizar operaciones sobre las bases de datos y las tablas, ya que se puede consultar utilizando sentencias SELECT. Las operaciones que afectan a la estructura de las bases de datos, como crear bases de datos, tablas o índices, actualizan estos archivos de metadata para reflejar los cambios en el sistema. Las funciones en el código se encargan de manipular y consultar esta información cuando se realizan operaciones sobre las bases de datos.

Punto 002 y 003

CREATE DATABASE <database-name>: Esta sentencia permite crear una nueva base de datos, la cual se modela como una carpeta en el sistema de archivos. El código utiliza el método **CreateDatabase** para crear la carpeta correspondiente y registrar la base de datos en el archivo **SystemDatabases.txt**, que actúa como un catálogo del sistema donde se almacena la lista de las bases de datos existentes.

SET DATABASE <database-name>: Esta sentencia establece el contexto de la base de datos en el cliente, permitiendo que las siguientes operaciones SQL sean ejecutadas en la base de datos seleccionada. El código verifica si la base de datos existe mediante el método **DatabaseExists**, validando su presencia en **SystemDatabases.txt**. Si la base de datos existe, el servidor confirma el cambio y el cliente establece el contexto, almacenando el nombre de la base de datos seleccionada en la variable **currentDatabase** para las próximas consultas SQL.

Punto 004 y 005

CREATE TABLE <table-name> AS (column-definition): Esta sentencia permite la creación de una tabla en una base de datos específica. El código usa el método **CreateTable** para procesar esta sentencia. Este método valida si ya hay una base de datos establecida en el contexto mediante **currentDatabase**. Luego, analiza la definición de las columnas para asegurarse de que los tipos de datos sean válidos (por ejemplo, **INTEGER**, **DOUBLE**, **VARCHAR(length)**, **DATETIME**) y aplica las restricciones de nulabilidad (**NULL** o **NOT NULL**). Si todo es correcto, se crea un archivo para la tabla dentro de la carpeta de la base de datos y la definición de la tabla se registra en el archivo **SystemTables.txt** del catálogo del sistema.

DROP TABLE <table-name>: Esta sentencia elimina una tabla de una base de datos. El método **DropTable** se encarga de verificar si la tabla existe y si está vacía, es decir, que no contenga datos. Si estas condiciones se cumplen, la tabla se elimina del sistema de archivos y también se actualiza el archivo **SystemTables.txt** para eliminar la referencia a la tabla. Además, cualquier índice asociado a la tabla también se elimina actualizando el archivo **SystemIndexes.txt**.

Punto 006

La sentencia **CREATE INDEX <index-name> ON <table-name>(<column-name>) OF TYPE <type>** permite crear un índice sobre una columna específica de una tabla. El código implementa esta funcionalidad en el método **CreateIndex**. Primero, se verifica si la tabla y la columna especificadas existen, y luego se comprueba que no existan valores repetidos en la columna que se desea indexar. Esto es importante porque, una vez creado el índice, no se permitirán valores duplicados en la columna indexada.

Existen dos tipos de índices que pueden crearse:

- **BTREE**: Implementa un árbol B para organizar los datos de manera eficiente en memoria, lo que facilita la búsqueda rápida de los registros.
- **BST (Binary Search Tree)**: Implementa un árbol binario de búsqueda para organizar los datos de la columna de manera jerárquica.

Una vez que se crea un índice, este se registra en el archivo **SystemIndexes.txt**, parte del catálogo del sistema. Los índices son recreados en memoria cuando el servidor se inicia, basándose en la información del catálogo. Esto asegura que los índices existentes estén disponibles después de reiniciar el servidor.

Punto 007

La sentencia **SELECT** permite seleccionar filas de una tabla específica, con opciones para especificar columnas, aplicar filtros con **WHERE**, y ordenar los resultados con **ORDER BY**. En el código, esta funcionalidad está implementada en el método **SelectFromTable**. Este método primero valida si la tabla existe y luego procesa la consulta. Si se especifica una cláusula **WHERE**, se evalúan las condiciones usando operadores como **>, <, =, like,** o **not.** Si hay un índice disponible para la columna filtrada, el código utiliza ese índice para acelerar la búsqueda. En caso contrario, se realiza una búsqueda secuencial en la tabla.

Además, si se especifica un **ORDER BY**, el código usa el algoritmo de **QuickSort** para ordenar los resultados en orden ascendente o descendente, según lo indicado. Finalmente, el resultado de la consulta se presenta en la interfaz del cliente como una tabla, facilitando la lectura de los datos consultados.

Punto 008

La sentencia **UPDATE <table-name> SET <column-name> = <new-value> [WHERE where-statement]** permite actualizar las filas de una tabla que cumplan con una condición **WHERE**. En el código, esta funcionalidad está implementada en el método **UpdateTable**. El proceso comienza validando la existencia de la tabla y de la columna a actualizar. Si se proporciona una cláusula **WHERE**, el código filtra las filas que cumplan la condición especificada utilizando operadores como >, <, =, etc. Si hay un índice en la columna mencionada en el **WHERE**, el código lo utiliza para optimizar la búsqueda. Si no, realiza una búsqueda secuencial.

Luego, se actualizan los valores de la columna especificada. Si la columna actualizada tiene un índice asociado, el índice también se actualiza para reflejar los nuevos valores, utilizando el método **UpdateIndexesAfterUpdate** para asegurar la coherencia entre los datos y los índices. Si no se especifica un **WHERE**, todas las filas de la tabla son actualizadas con el nuevo valor proporcionado.

Punto 009

La sentencia **DELETE FROM <table-name> [WHERE where-statement]** elimina filas de una tabla que cumplan con la condición especificada en la cláusula **WHERE**. En el código, esta funcionalidad está implementada de manera similar a la sentencia **UPDATE**. Primero, si se especifica un **WHERE**, se busca la fila o filas que coincidan con la condición, utilizando índices si están disponibles para optimizar la búsqueda. Si no hay índices disponibles, se realiza una búsqueda secuencial en la tabla.

Cuando se elimina una fila, el código también se asegura de actualizar cualquier índice asociado a la tabla o columna afectada, de modo que la eliminación refleje los cambios en la estructura del índice. Si no se especifica un **WHERE**, se eliminan todas las filas de la tabla. Además, si hay índices en las columnas afectadas, estos también se actualizan o eliminan según corresponda para mantener la consistencia de los datos.

Punto 010

La sentencia **INSERT INTO <table-name> VALUES(<values>, <values>...)** inserta valores en una tabla en el orden en que las columnas fueron definidas al momento de crear la tabla. En el código, esta funcionalidad está implementada en el método InsertIntoTable, el cual primero valida que los tipos de datos de los valores sean correctos para cada columna. Si alguna columna es de tipo **DATETIME**, el valor se proporciona como una cadena (String), pero el código internamente convierte (parsea) este valor a un formato de fecha y hora (DateTime).

Además, el código verifica que no se ingresen valores duplicados en columnas que tienen un índice asociado, asegurando la integridad de los datos. Si no hay problemas, los valores son añadidos a la tabla y cualquier índice que esté vinculado a las columnas afectadas se actualiza para reflejar los nuevos datos.

Diagrama UML