Instituto Tecnológico de Costa Rica

Ingeniería en computadores

Curso: Algoritmos y Estructuras de Datos I

Proyecto II TinySQLDb

II Semestre 2024

Integrantes:
Alvarez Chanto Sergio Daniel
2024134332
Altamirano Cordero Esteban Andrés
2024222350

1.Introducción

El objetivo principal del proyecto es crear y diseñar un motor de bases de datos relacional sencillo llamado TinySQLDb, este proyecto está basado en parte de SQL, como en el uso de la interfaz de usuario y en la ejecución de las sentencias SQL pero internamente no utiliza una biblioteca o motor SQL ya existente, en lugar de eso se debe implementar el procesamiento de consultas SQL y el manejo de los datos en el servidor.

Además, el proyecto utiliza el lenguaje de programación C# siguiendo el paradigma de orientación a objetos. Además, se apoya en PowerShell para proporcionar una interfaz de cliente, lo que permite ejecutar consultas SQL de manera interactiva.

Tabla de contenidos

2.Descripción del problema

El problema consiste en la implementación de un sistema de bases de datos relacional sencilla que permita a los usuarios realizar operaciones en la base de datos como crear bases de datos, crear tabla, la inserción de registros, actualización y consultas usando un subconjunto del lenguaje SQL, también el sistema debe de ser capaz de gestionar los datos almacenados en archivos binarios.

Además, el sistema debe soportar la creación de índices para mejorar la eficiencia en las consultas y proveer una interfaz amigable para la ejecución de sentencias SQL mediante

PowerShell. Los resultados de las consultas deben mostrarse en formato de tabla en la terminal.

3.Descripción de la solución

Requerimiento 000: Cliente de Powershell

Implementación

Se creó un script de PowerShell (Tinysqlclient.ps1) que actúa como un cliente para interactuar con el servidor de bases de datos. Utiliza la función "Send-SQLCommand" para enviar consultas SQL al servidor, y la función "Receive-Message" para recibir respuestas. También implementa un bucle que permite al usuario ingresar comandos SQL interactivos.

Alternativas Consideradas

Una alternativa fue usar un cliente en C# en vez de PowerShell pero PowerShell permite una integración más directa en la terminal y una forma más sencilla de manipular la entrada y salida.

Limitaciones

Powershell no es muy práctico para manejar grandes cantidades de datos como lo haría una aplicación en C# por ejemplo, y podría tener limitaciones en el rendimiento de operaciones muy largas o en grandes bases de datos.

Problemas Encontrados

Uno de los retos pudo haber sido conectarse al servidor y cerrar la conexión de red correctamente después de cada comando para evitar conexiones abiertas además de comprender la sintaxis de PowerShell

Requerimiento 001: Base de datos como carpeta en el sistema de archivos

Implementación

Cada base de datos es implementada como una carpeta en el sistema de archivos, y cada tabla es representada por archivos binarios dentro de esa carpeta. La metadata está almacenada en el "SystemCatalog", que es una carpeta que contiene archivos con información sobre las bases de datos y tablas.

Alternativas Consideradas

Una alternativa era usar una base de datos en memoria el cual es un sistema de gestión de bases de datos que almacena datos principalmente en la memoria RAM en lugar de en discos duros, sin embargo se utilizó el sistema de archivos directamente porque permite que los datos se guarden de forma duradera en un medio que no se pierde al apagar o reiniciar la aplicación.

Limitaciones

El manejo de archivos binarios podría limitar la escalabilidad y hacer que las operaciones de entrada/salida sean más lentas en comparación con motores de bases de datos más avanzados que usan optimizaciones específicas para el almacenamiento como compresión de datos, caché, particionamiento, etc.

Problemas Encontrados

En sistemas operativos con políticas de seguridad más estrictas, el programa podría tener problemas de acceso al intentar crear carpetas o archivos si no se ejecuta como administrador.

Requerimiento 002: CREATE DATABASE

Implementación

La base de datos se modela como una carpeta dentro del sistema de archivos. El método CreateDatabase verifica si la carpeta (base de datos) ya existe y de no ser así la crea, también se asegura de que el catálogo del sistema esté inicializado.

```
public OperationResult CreateDatabase(string databaseName)
{
    var databasePath = $@"{DataPath}\{databaseName}";

    if (Directory.Exists(databasePath))
    {
        Console.WriteLine($"Error: La base de datos {databaseName} ya existe.");
        return new OperationResult(OperationStatus.Error, "Database already exists.");
    }

    try
    {
        // Crear el directorio para la base de datos
        Directory.CreateDirectory(databasePath);
        Console.WriteLine($"Base de datos {databaseName} creada exitosamente en {databasePath}.");
        return new OperationResult(OperationStatus.Success, "Database created successfully.");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error al crear la base de datos {databaseName}: {ex.Message}");
        return new OperationResult(OperationStatus.Error, $"Failed to create database: {ex.Message}");
    }
}
```

Alternativas Consideradas

Se podría haber utilizado una estructura en memoria en lugar del sistema de archivos pero esto hubiera complicado la persistencia de datos en caso de un reinicio del sistema.

Limitaciones

Usar el sistema de archivos limita la escalabilidad y el manejo de errores ya que se depende de las capacidades del sistema operativo para gestionar directorios y permisos.

Problemas Encontrados

Un problema fue garantizar que la base de datos no existiera previamente ya que esto generaba un conflicto, por eso se implementó un método que valida la existencia de la base de datos.

Requerimiento 003: SET DATABASE

Implementación

El comando "SET DATABASE" establece el contexto de base de datos dentro del cliente powershell, el método SetDatabase implementado en Store.cs válida si la carpeta de la base de datos existe y de ser así establece el contexto para las operaciones posteriores.

```
public OperationResult SetDatabase(string databaseName)
{
    var databasePath = $@"{DataPath}\{databaseName}";

    if (Directory.Exists(databasePath))
    {
        Console.WriteLine($"Base de datos {databaseName} establecida exitosamente.");
        return new OperationResult(OperationStatus.Success, "Database exists.");
    }
    else
    {
        Console.WriteLine($"Error: La base de datos {databaseName} no existe.");
        return new OperationResult(OperationStatus.Error, "Database does not exist.");
    }
}
```

Alternativas Consideradas

Manejar el contexto en el servidor en lugar del cliente, pero se decidió que fuese el cliente quien lo manejara para simplificar la implementación.

Limitaciones

El contexto es manejado por cada cliente de manera independiente, lo que puede complicar la sincronización entre múltiples usuarios.

Problemas Encontrados

Asegurar que las consultas posteriores respeten el contexto seleccionado.

Requerimiento 004: CREATE TABLE

Implementación

El método CreateTable toma el nombre de la base de datos, el nombre de la tabla y las columnas especificadas en la consulta SQL y genera un archivo binario que representa dicha tabla en la carpeta correspondiente a la base de datos También se actualiza el catálogo del sistema con la nueva información sobre la tabla creada para que pueda ser consultada posteriormente

```
public OperationResult CreateTable(string databaseName, string tableName, List<Column> columns)
{
    var databasePath = $@"{DataPath}\{databaseName}";
    if (!Directory.Exists(databasePath))
    {
        return new OperationResult(OperationStatus.Error, "Database does not exist.");
    }

    Console.WriteLine($"Creando tabla {tableName} en la base de datos {databaseName} con {columns.Count} columnas.");

    // Verifica si hay columnas definidas antes de continuar
    if (columns.Count == 0)
    {
        return new OperationResult(OperationStatus.Error, "No se han definido columnas para la tabla.");
    }

    // Crea la tabla: Solo crea un archivo vacío como indicador de la existencia de la tabla
    var tablePath = $@"{databasePath}\{tableName}.table";
    if (File.Exists(tablePath))
    {
        return new OperationResult(OperationStatus.Error, "Table already exists.");
    }

    using (FileStream stream = File.Open(tablePath, FileMode.Create))
    {
        Console.WriteLine($"Archivo de la tabla {tableName} creado.");
    }

    // Actualiza el catálogo del sistema con la nueva tabla y sus columnas
    UpdateSystemCatalog(databaseName, tableName, columns);

    // Aquí agregas el código para verificar las columnas
    var createdColumns = GetTableDefinition(databaseName, tableName);
    Console.WriteLine($"Número de columnas en la tabla creada: {createdColumns.Count}");
    foreach (var column in createdColumns)
    {
        Console.WriteLine($"Columna: {column.Name}, Tipo: {column.Type}, Tamaño: {column.Size}");
    }

    return new OperationResult(OperationStatus.Success, "Table created successfully.");
}
```

Alternativas Consideradas

Usar formatos de archivo como JSON o CSV, pero se optó por binarios para mejor eficiencia(Velocidad de acceso, tamaño en disco).

Limitaciones

Solo soporta tipos de datos básicos y no admite relaciones entre tablas ni restricciones avanzadas.

Problemas Encontrados

La necesidad de validar tipos de datos y manejar adecuadamente las cadenas VARCHAR puede hacer que el código sea más complejo.

Requerimiento 005: DROP TABLE

Implementación

Se encarga de eliminar una tabla siempre que esté vacía. Primero, valida si el archivo de la tabla existe. Si la tabla existe, abre el archivo en modo lectura y verifica si su tamaño es mayor a cero, lo que indica si tiene registros. Si la tabla está vacía, procede a eliminar el archivo correspondiente. En caso de error al eliminar, maneja la excepción y devuelve un mensaje de error.

```
internal class DropTable
{
    private readonly string _databaseName;
    private readonly string _tableName;

    1 referencia
    public DropTable(string databaseName, string tableName)
    {
        _databaseName = databaseName;
        _tableName = tableName;
    }

    1 referencia
    public OperationResult Execute()
    {
        var dataPath = Store.GetInstance().GetDataPath();
        var tablePath = $@"{dataPath}\{_databaseName}\{_tableName}.table";
        Console.WriteLine($"Attempting to drop table: {_tableName} in database: {_databaseName}");

        if (!File.Exists(tablePath))
        {
            Console.WriteLine("Error: Table does not exist.");
            return new OperationResult(OperationStatus.Error, "Table does not exist.");
        }

        // Check if the table is empty
        using (FileStream stream = File.Open(tablePath, FileMode.Open))
        {
            if (stream.Length > 0)
            {
                Console.WriteLine("Error: Table is not empty. Cannot drop a non-empty table.");
                return new OperationResult(OperationStatus.Error, "Table is not empty. Cannot drop a non-empty table.");
            }
        }

        // Delete the table file
        try
        {
            File.Delete(tablePath);
            Console.WriteLine($"Table {_tableName} successfully dropped from database {_databaseName}.");
            return new OperationResult(OperationStatus.Success, "Table dropped successfully.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error deleting table {_tableName}: {ex.Message}");
            return new OperationResult(OperationStatus.Error, "Error deleting table: " + ex.Message);
        }
    }
}
```

Alternativas Consideradas

Manejar el contexto en el servidor en lugar del cliente, pero se decidió que fuese el cliente quien lo manejara para simplificar la implementación.

Limitaciones

El contexto es manejado por cada cliente de manera independiente, lo que puede complicar la sincronización entre múltiples usuarios.

Problemas Encontrados

Requerimiento 006: CREATE INDEX

Implementación

Los índices se crean en memoria utilizando árboles B (BTree) o binarios (BST). Se registra la existencia del índice en el System Catalog y se utiliza para búsquedas optimizadas.

```
internal class CreateIndex
{
    private readonly string _databaseName;
    private readonly string _tableName;
    private readonly string _columnName;
    private readonly string _indexName;
    private readonly string _indexType; // BTREE o BST

    1 referencia
    public CreateIndex(string databaseName, string tableName, string columnName, string indexName, string indexType)
    {
        _databaseName = databaseName;
        _tableName = tableName;
        _columnName = columnName;
        _indexName = indexName;
        _indexType = indexType;
    }

    1 referencia
    public OperationResult Execute()
    {
        var store = Store.GetInstance();

        // Obtener la definición de las columnas de la tabla
        var columns = store.GetTableDefinition(_databaseName, _tableName);
        var column = columns.Find(c => c.Name == _columnName);

        if (column == null)
        {
            Console.WriteLine("Error: La columna especificada no existe en la tabla.");
            return new OperationResult(OperationStatus.Error, "La columna especificada no existe en la tabla.");
        }

        // Verificar si la columna ya tiene un índice
        if (store.IndexExists(_databaseName, _tableName, _columnName))
        {
            Console.WriteLine("Error: Ya existe un índice en esta columna.");
            return new OperationResult(OperationStatus.Error, "Ya existe un índice en esta columna.");
        }

        // Verificar si la columna tiene valores repetidos antes de crear el índice
        if (store.HasDuplicateValues(_databaseName, _tableName, _columnName))
        {
            Console.WriteLine("Error: No se puede crear un índice sobre una columna con valores repetidos.");
            return new OperationResult(OperationStatus.Error, "No se puede crear un índice sobre una columna con valores repetidos.");
        }

        // Crear el índice en memoria y guardar la referencia en el catálogo del sistema
        var indexPath = $@"{store.GetDataPath()}\{_databaseName}\{_indexName}.index";
        using (FileStream stream = File.Open(indexPath, FileMode.Create))
        {
            Console.WriteLine($"Creando índice {_indexName} de tipo {_indexType} en la columna {_columnName} de la tabla {_tableName}.");
            // Aquí deberías implementar la lógica para crear el árbol B o el BST, y almacenar la referencia.
            // Por simplicidad, esto solo crea un archivo vacío.
        }

        // Actualizar el catálogo del sistema con la información del índice creado
        store.UpdateSystemCatalogWithIndex(_databaseName, _tableName, _columnName, _indexName, _indexType);

        return new OperationResult(OperationStatus.Success, "Índice creado con éxito.");
    }
}
```

Alternativas Consideradas

Utilizar tablas hash, pero se prefirieron los árboles B por su eficacia en búsquedas y ordenamiento.

Limitaciones

Solo se permite un índice por columna y debe recrearse en memoria cada vez que el servidor se reinicia.

Problemas Encontrados

La validación de que no hubieran duplicados en las columnas.

Requerimiento 007: SELECT

Implementación

El comando "SELECT" permite seleccionar filas de una tabla, filtrar los resultados mediante condiciones WHERE y ordenándolos si es necesario con ORDER BY, si la consulta se refiere a una columna que tiene un índice se utiliza este índice para mejorar la velocidad de búsqueda.

Alternativas Consideradas

La optimización del plan de ejecución de consultas para evaluar varias formas de ejecutar una consulta y elegir la más eficiente en función dependiendo de los factores como el tamaño de las tablas, los índices y las condiciones de las consultas.

Limitaciones

Solo soporta operaciones básicas de selección, sin uniones entre tablas ni consultas complejas.

Problemas Encontrados

Evaluar correctamente las condiciones WHERE para distintos tipos de datos

Requerimiento 008: UPDATE

Implementación

Actualiza registros que cumplen con la condición WHERE y reescribe el archivo de la tabla. Si se actualiza una columna con índice, este también se modifica.

Alternativas Consideradas

La opción de realizar actualizaciones en memoria y escribir en disco periódicamente es una alternativa porque puede mejorar el rendimiento del sistema.

Limitaciones

El rendimiento es bajo al reescribir el archivo completo para actualizar un dato

Problemas Encontrados

Actualizar los índices después de modificar los datos

Requerimiento 009: DELETE

Implementación

Elimina los registros que cumplan con la condición WHERE reescribiendo el archivo sin esas filas.

Alternativas Consideradas

Implementar un "borrado lógico" es una alternativa porque en vez de eliminar físicamente los registros del archivo, se marcarían como "eliminados" sin eliminarlos realmente.

Limitaciones

Es ineficiente al tener que reescribir el archivo completo como en el UPDATE.

Problemas Encontrados

Actualizar los índices y manejar correctamente los permisos del sistema al eliminar registros

Requerimiento 010: INSERT INTO

Implementación

El método INSERT INTO recibe el nombre de la tabla y de los valores que se van a insertar en el archivo de la tabla, valida que los tipos de datos sean correctos y actualiza los índices correspondientes.

Alternativas Consideradas

Mantener los registros en memoria y escribir en disco periódicamente, pero se prefirió la inserción inmediata para asegurar la persistencia.

Limitaciones

El rendimiento puede verse afectado por la escritura directa en tablas grandes.

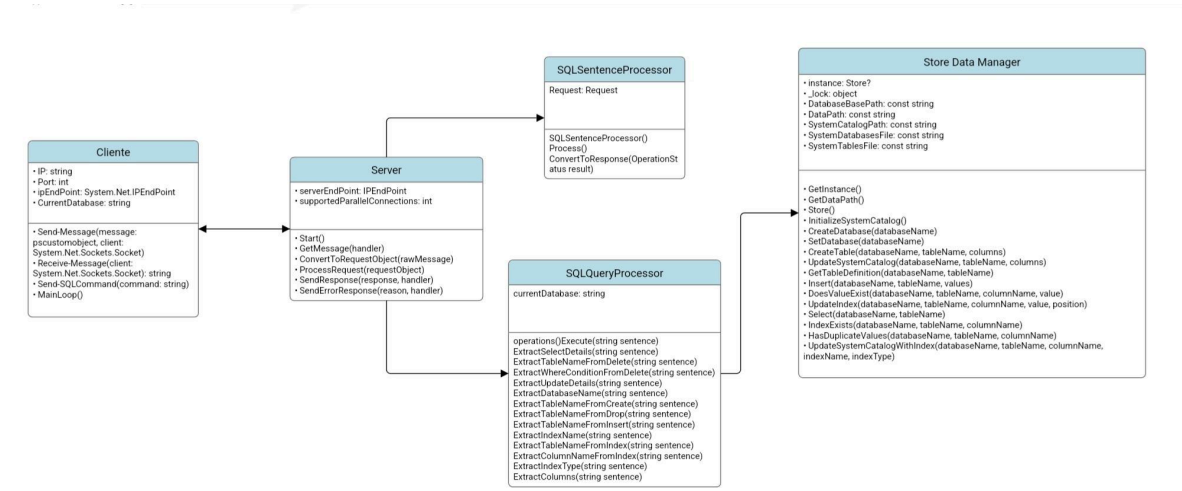
Problemas Encontrados

La validación de tipos de datos y el manejo de índices al insertar nuevos registros

4.Diseño general

Patrón Factory Method, este patrón se aplica en la creación de las operaciones SQL como Insert, Delete, CreateTable, Select, porque permite que las clases utilicen objetos sin tener que

saber cómo se crean.



5.Bibliografía:

Stack Overflow: K. R. Simas, “Binary Search Tree in C# Implementation,” Stack Overflow, 2011.https://stackoverflow.com/questions/10366402/binary-search-tree-in-c-sharp-implementation

DelftStack: “How to Create a Binary Search Tree in .NET 4.0,” DelftStack, 2020.https://www.delftstack.com/es/howto/csharp/binary-search-tree-in-.net-4.0/

Microsoft Docs: Microsoft, “How to Use Docs,” Microsoft Docs, 2023.https://learn.microsoft.com/en-us/powershell/scripting/how-to-use-docs?view=powershell-7.4

YouTube: “PowerShell 7 Tutorial 3: Fundamentals for Beginners - Introduction to PowerShell Concepts,” YouTube, 2020.https://www.youtube.com/watch?v=pr9iVsKAiuc

Microsoft Docs: Microsoft, “Installing PowerShell on Windows,” Microsoft Docs, 2023.https://learn.microsoft.com/en-us/powershell/scripting/install/installing-powershell-on-windows?view=powershell-7.4

AWS: “What is SQL?,” Amazon Web Services, 2023. https://aws.amazon.com/es/what-is/sql/