TEC | Tecnológico de Costa Rica

TinySQLDb, 2024, 1–4

https://github.com/lVoidi/TinySQLdb.git

TinySQLDb

David Garcia Cruz[1,*,†] and Rodrigo Arce Bastos[2,*,†]

[1]Tecnologico de Costa Rica

[*]r.arce.1@estudiantec.cr [†]d.garcia.1@estudiantec.cr

Introducción

Este documento presenta el desarrollo de un programa que implementa un motor de bases de datos de forma sencilla utilizando Powershell 7 y Csharp para cumplir con los requerimientos del proyecto, abordando un desafio complejo de ingeniería. El objetivo del proyecto es familiarizar a los estudiantes con el funcionamiento sencillo de una base de datos, además de emplear el paradigma orientado a objetos (POO) a la hora de diseñar e implementar TinySQLDb. Para esto se debe investigar el funcionamiento de una base de datos real y buscar los componentes básicos de una base para poderlos implementar en la solucion del proyecto. El plan de solucion implica análisis de los requerimientos necesarios, así como definir el diseño del sistema, el desarrollo del software y para verificar el funcionamiento realizar pruebas y validaciones, optimizando el código al máximo, además para mantener un control de versiones adecuado se utilizara la herramienta Git.

Tabla de Contenidos

Descripción del problema

El problema consiste en diseñar e implementar una base de datos relacional sencilla, en la cual se debe tener un cliente el cual es un modulo de Powershell 7 que le permite al usuario o ingresar consultas en un subconjunto del lenguaje SQL mediante línea de comandos. Envía la consulta al motor de bases de datos mediante socket y renderiza el resultado en la consola en formato de tabla. Para los comando requeridos se tienen los principales de un motor de base de datos como por ejemplo CREATE DATABASE <database-name>, SET DATABASE <database-name>, CREATE TABLE <table-name> AS (column-definition);, etc. Además del cliente se debe tener un servidor el cual está compuesto por distintas capas la primera siendo API interface, esta capa posee la lógica para escuchar los mensajes entrantes por el socket y para enviar la respuesta de vuelta al cliente. La segunda capa es Query processing en esta capa El API interface llama métodos de esta capa para procesar la consulta. La capa de Query processing, valida la sintaxis de la consulta y verifica que semánticamente sea correcta, es decir que tablas, columnas, tipos de datos y cualquier otro detalle sea correcto. La última capa se denomina Stored Data Manager en esta capa se accede a los archivos correspondientes a cada tabla de cada base de datos y a los índices asociados a cada una. Todo lo mencionado anteriormente es lo alusivo al problema por resolver utilizando POO y algoritmos de estructuras de ordenamiento como por ejemplo árboles.

Compiled on: October 11, 2024.
Draft manuscript prepared by the author.

Solución al problema

Para la solucion al problema dado se utilizo el lenguaje C sharp, y ademas se utilizo Git para el manejo de versiones y la herramienta de las branches que Git provee para evitar conflictos de versiones entre los dos colaboradores del proyecto. A continuación se procederá a detallar las implementaciones, limitaciones y problemas encontrados para cada uno de los requerimientos del proyecto.

Cliente y conexión con el servidor

Para este requerimiento en el cual el cliente debe ser un modulo de powershell se cre un script que implementa la funcionalidad requerida. El script define una funcion principal llamada Send-SQLCommand que realiza el siguiente funcionamiento, primero lee el contenido del archivo SQL especificado, establece una conexion con el servidor mediante el uso de sockets, luego envía el comando SQL al servidor y este recibe y procesa la respuesta. El script implementa funciones auxiliares como Send-Message y Receive-Message para manejar la comunicación, la función principal Send-SQLCommand convierte el comando SQL en un objeto JSON antes de enviarlo al servidor, y luego convierte la respuesta JSON del servidor de vuelta a un objeto PowerShell. El problema encontrado para este fue el uso de los sockets, el cual a pesar de tener una base otorgada por el profesor fue necesario investigar un poco y hacer modificaciones para poder entender de manera correcta su funcionamiento.

System Catalog

Para resolver este requerimiento relacionado a la gestión de los datos, se hizo una clase llamada Data la cual se encarga de la creacion, lectura, actualización y eliminación de bases de datos y tables, además se utilizan archivos JSON para guardar y cargar los datos de las tablas, mediante las funciones SaveTable y LoadTable. Tambien se implementan los tipos de índices, de estos profundizaremos mas adelante, los cuales optimizan las búsquedas en las tablas. La clase SQLQueryProcessor se encarga de interpretar y ejecutar las consultas SQL interactuando con las clase Data. Por ultimo se tiene el servidor para la conexión con el cliente y el script de powershell que permite enviar comandos al servidor y recibir respuestas. La dificultad presentada con este requerimiento se dio con guardar las tablas y los datos, ya que hubo que investigar como guardar y cargar información en un JSON, esta fue la mayor complicación para este requerimiento.

Comando SQL

En esta sección se describirá el funcionamiento y la implementación de los comandos utilizados.

CREATE DATABASE

Para este comando en la clase SQLQueryProcessor, el método Parse identifica el comando "CREATE DATABASE" utilizando una expresión regular, si la expresión coincide se extrae el nombre de la base de datos, posteriormente se llama al metodo CreateDatabase de la instancia data en la clase Data y en esta clase se crea la base, primero se verifica que el nombre de la base de datos no este vacio, crea una nueva instancia de DataPath con el nombre de la base de datos y se crea el directorio, además se actualiza el System Catalog. Para saber si la creacion fue exitosa el sistema devuelve un mensaje de éxito indicando que la base de datos ha sido creada. Para este metodo no hubo mayor problema, como mucho fue guardar la tabla en la clase Data.

SET DATABASE

Para este metodo primero el metodo Parse identifica que la expresión este bien escrita, posteriormente se llama al metodo SetDatabase de la clase Data y se valida la existencia de la base de datos, luego se verifica que la base de datos exista y se estable el contexto actualizando la propiedad database. Una vez se verifica el servidor envía una respuesta al cliente mostrando que la operación fue exitosa.

CREATE TABLE

Primero el metodo Parse identifica que la expresión este bien escrita, luego se extraer el nombre de la tabla y las definiciones de las columnas, se separan las definiciones de columnas y se procesan de manera individual. Luego se llama al metodo CreateTableAs de la clase Data el cual procesa la definicion de columna extrayendo el nombre, tipo y tamaño, crea los objetos Field para cada columna y los agrega en una lista. Ahora se crea una nueva instancia de Table con el nombre y los encabezados definidos. Por ultimo agrega la nueva tabla a la lista de tablas en la base de datos. El problema mas grande con este requerimiento fue el guardado de las tablas y la actualizacion de SystemCatalog así como el manejo que había que darle a los parámetros establecidos para la tabla.

DROP TABLE

Primero se identifica que el comando este bien escrito en la clase SQLQueryProcessor y una vez esto se verifique el metodo DropTable hace lo siguiente; verifica que la tabla exista, comprueba si la tabla esta vacia y si esto cumple procede a eliminar la tabla de la base de datos, l método Path.Drop(tableName) se encarga de eliminar el archivo físico que representa la tabla en el sistema de archivos y se actualiza el System Catalog. Lo mas complicado para este comando fue el metodo de como eliminar la tabla accediendo a la memoria del dispositivo.

CREATE INDEX

El comando CREATE INDEX crea un índice en una columna de una tabla. El sistema verifica la existencia de la tabla y columna, y que no haya valores repetidos. Se crea una estructura de datos en memoria (árbol B para BTREE, árbol binario de búsqueda para BST) que contiene los valores de la columna y referencias a los registros completos. El índice se registra en el SystemCatalog para persistencia. Tras un reinicio del servidor, los índices se recrean en memoria basándose en esta información. El sistema mantiene el índice actualizado con cada operación en la tabla y previene la inserción de valores duplicados en la columna indexada. Este proceso lo que busca es agilizar la busqueda de los datos en la base de datos mediante el uso de arboles e indices. Este requerimiento tuvo muchos problemas para poder manejar bien los arboles, hubieron muchas cambios en los dos arboles, por mal manejo de instancias y funciones que estaban mal empleadas, además las pruebas de velocidad aveces no concordaban de manera correcta.

SELECT

El comando SELECT recupera datos de una tabla. El parser analiza la sentencia para identificar columnas, tabla, condiciones WHERE y ORDER BY. El método Select verifica la existencia de la tabla y procesa la consulta. Para cláusulas WHERE, usa índices si existen utilizando los métodos search dependiendo del árbol en las clase IndexBTree o IndexBSTree. Las condiciones se evalúan usando operadores como >, <, =, LIKE, o NOT. Si se especifica ORDER BY, se usa Quicksort para ordenar los resultados. Finalmente, los datos se formatean en una estructura tabular y se envían al cliente. Este proceso permite consultas eficientes, aprovechando índices cuando están disponibles y ofreciendo ordenamiento cuando se solicita. El problemas mas grande para este comando fue la implementación de los arboles de manera que el metodo select funcionara bien y mostrara de manera correcta los datos al cliente.

UPDATE
El comando UPDATE modifica datos en una tabla específica. El parser SQL analiza la sentencia para identificar la tabla, columnas a actualizar, nuevos valores y condiciones WHERE. El método Update en la clase Data verifica la existencia de la tabla y columnas. Para cláusulas WHERE, el sistema usa índices si existen. Se actualizan los valores de las columnas especificadas en cada fila que cumple la condición. Si se actualizan columnas con índices, estos también se actualizan, eliminando entradas antiguas y agregando nuevas. El proceso maneja casos especiales, como evitar duplicados en índices que no los permiten. El sistema confirma la cantidad de filas actualizadas y actualiza metadatos relevantes en el SystemCatalog. Un problema que presento este comando es que al inicio las tablas o la informacion no se estaba actualizando de manera correcta.

El comando DELETE elimina filas de una tabla específica. El parser SQL analiza la sentencia para identificar la tabla y las condiciones WHERE. El método Delete en la clase Data verifica la existencia de la tabla. Para cláusulas WHERE, el sistema usa índices si existen, o realiza búsqueda secuencial. Se eliminan las filas que cumplen la condición. Si la tabla tiene índices, estos se actualizan eliminando las entradas correspondientes a las filas borradas. El proceso maneja la reorganización del espacio en el archivo de la tabla. El sistema confirma la cantidad de filas eliminadas y actualiza los metadatos.

INSERT INTO
El comando INSERT INTO añade nuevas filas a una tabla. El parser SQL analiza la sentencia para identificar la tabla y los valores a insertar. El método Insert en la clase Data verifica la existencia de la tabla y la correspondencia entre valores y columnas. El sistema valida que los tipos de datos coincidan con los definidos para cada columna, convirtiendo strings a DateTime cuando es necesario. Luego, crea un nuevo registro y lo añade al archivo de la tabla. Si existen índices, se actualizan añadiendo entradas que apunten al nuevo registro. El proceso maneja casos especiales, como evitar duplicados en índices únicos. Para este requerimiento se tuvo la complicación de que habían errores al intentar insertar la tabla generalmente los errores eran relacionados con los índices y la actualización de los datos en el SystemCatalog.

Diagrama UML

En la siguiente pagina se presenta el diagrama de clases para todo el proyecto cada clase con sus respectivos métodos y atributos. Las flechas con un rombo negro, representan una relación de composición, las flechas con un rombo vació representan una relación de agregación. Las lineas continuas con una flecha vacía representan herencia y la linea discontinua con la punta en V representa una relación de dependencia.

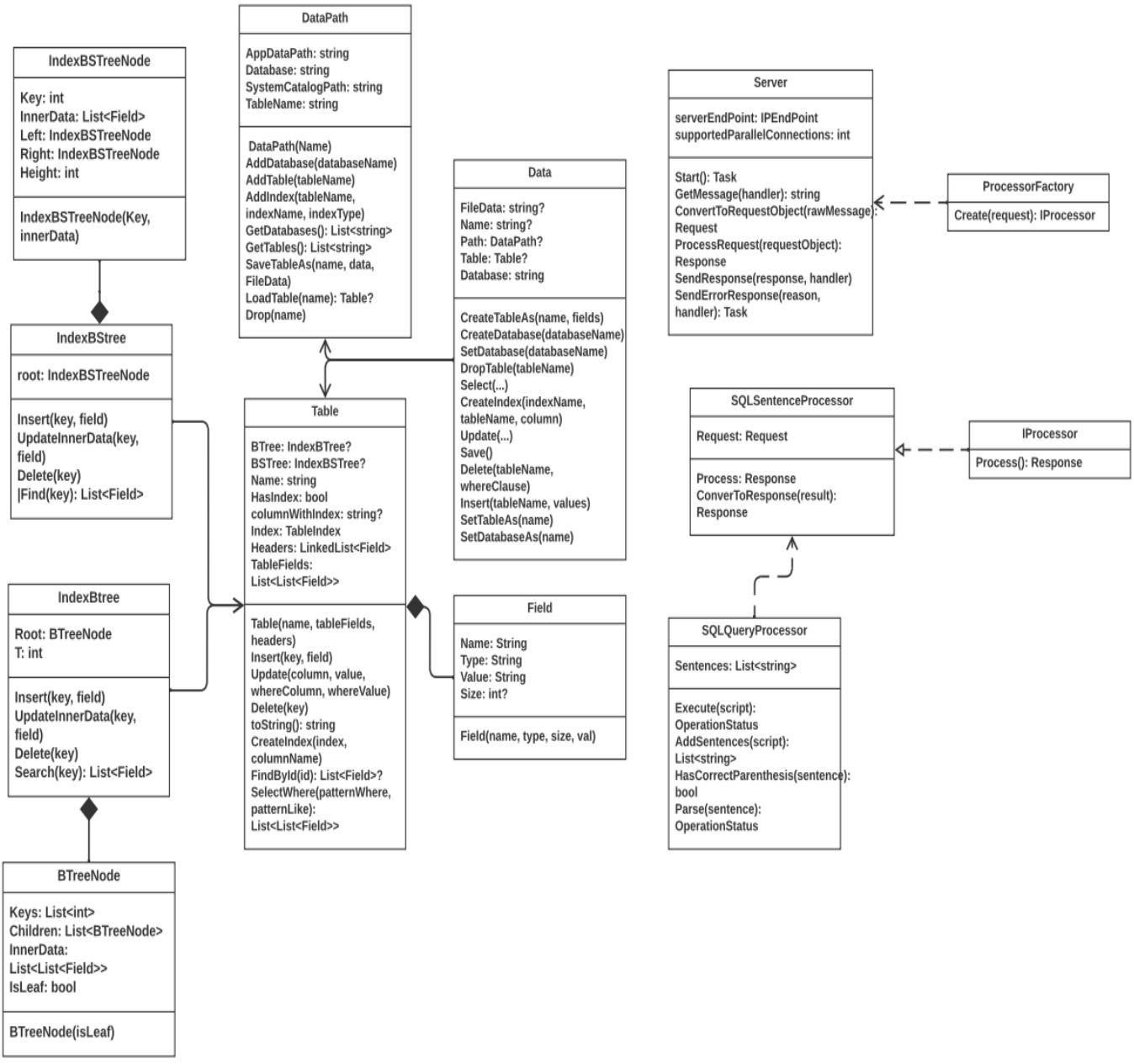

Figure 1. Diagrama UML del proyecto