PROYECTO II

TinySQLDb

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

OBJETIVOS

GENERAL

- Diseñar e implementar un motor de bases de datos relacional sencillo para familiarizar a los estudiantes con el funcionamiento de este.

ESPECÍFICOS

- Implementar algunos componentes básicos de un motor de bases de datos relacional
- Investigar cómo funcionan los motores de bases de datos más conocidos en la práctica
- Diseñar e implementar la solución en el lenguaje de programación C#
- Emplear el paradigma de orientación a objetos a la hora de diseñar e implementar TinySQLDb

REQUERIMIENTOS

Se recomienda instalar algún motor de bases de datos comercial como MySQL y tomar ideas de cómo funcionan y cuál es la experiencia del usuario al ejecutar sentencias SQL

El proyecto consiste en implementar un sistema administrador de bases de datos sencillo. Este sistema tiene los siguientes componentes:

1. **Cliente**: módulo de Powershell que le permite al usuario 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.

2. **Servidor**: es un sistema compuesto por las siguientes capas:
    a. *API interface*: esta capa contiene toda la lógica para escuchar los mensajes entrantes por el socket y para enviar la respuesta de vuelta al cliente. Toda la comunicación entre cliente y servidor es en formato JSON. El API interface es el único punto del sistema que manipula JSON.

b. *Query processing*: 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. **Para parsear las sentencias SQL, puede utilizar alguna biblioteca externa, siempre y cuando pueda adaptarla a las particularidades de lo soportado por TinySQL.**

c. *Stored Data Manager*: Esta capa accede los archivos correspondientes a cada tabla de cada base de datos y a los índices asociados a cada una.

En términos generales, cuando se ejecuta una sentencia, se sigue el siguiente procedimiento:

1. El cliente envía el request mediante un socket al API Interface
2. API interface procesa el mensaje y envía la sentencia SQL al query processor quitando cualquier elemento específico del protocolo de comunicación
3. Query processor valida que la sentencia sea correcta utilizando el system catalog (por ejemplo, que la base de datos exista, que la tabla exista, que las columnas sean válidas, entre otras) y coordina su ejecución apoyándose en Stored Data Manager
4. Stored Database Manager accede a los archivos que contienen los datos y realiza la consulta como tal.

A continuación, se detalla cada uno de los requerimientos

ID	DESCRIPCIÓN	PUNTOS
000	El cliente es un módulo de Powershell 7 que se carga en Powershell y que permite ingresar comandos para realizar operaciones sobre la base de datos. Se crea una función llamada *Execute-MyQuery* con los siguientes parámetros: - QueryFile: Indica un path donde se encuentra el archivo con sentencias SQL por ejecutar. - Port: puerto en el que el API interface escucha - IP: dirección IP en el que el API interface escucha Por ejemplo: Execute-MyQuery –Query .\Script.tinysql -Port 8000 –IP 10.0.0.2 Los resultados se muestran en la terminal en formato de tabla. Powershell como tal, provee muchas facilidades para formatear objetos en formato tabla, puede investigar y usar dichas funciones.	15

	Cada sentencia dentro del archivo se ejecuta una por una y se muestra el resultado de cada una. Por cada sentencia que se ejecuta se muestra el tiempo que se tardó el servidor en completarla. Cada sentencia del script se separa por punto y coma.	
001	Una base de datos se modela como una carpeta en el sistema de archivos. Archivos **binarios** dentro de esta carpeta corresponden a tablas de la base de datos. El system catalog es una carpeta con archivos **binarios** compartida para todas las bases de datos, que contiene metadata, es decir, datos sobre los datos. Almacena información como: 1. Listado de las bases de datos existentes (archivo SystemDatabases) 2. Tablas existentes en cada base de datos (archivo SystemTables) 3. Columnas de cada tabla (archivo SystemColumns) 4. Índices asociados a cada table (archivo SystemIndexes) Las tablas de System Catalog se pueden consultar con sentencias SELECT	10
002	Para crear una base de datos, la sentencia SQL es: CREATE DATABASE <database-name>	10
003	La sentencia: SET DATABASE <database-name> establece el contexto en el cliente. El servidor únicamente valida que la base de datos solicitada exista. El cliente al recibir la respuesta del server establece el contexto para las siguientes sentencias SQL	5
004	Para crear una tabla para en una base de datos específica, se usa la sentencia CREATE TABLE <table-name> AS (column-definition); column-definition: <column-name> type nullability constraint type: INTEGER | DOUBLE | VARCHAR(length) | DATETIME Integer y Double corresponde a los enteros y double tradicionales. Varchar es un string pero debe ser de tamaño fijo, por lo que se deberá especificar el tamaño máximo al crear la tabla Datetime es un formato fecha y hora.	10
005	DROP TABLE <table-name>	5

	Elimina una tabla de la base de datos establecida en el contexto, siempre y cuando la tabla esté vacía.	
006	CREATE INDEX <index-name> ON <table-name>(<column-name>) OF TYPE <type> Type: BTREE, BST Permite crear índices sobre una columna de la tabla. Para poder agregar a un índice a una tabla con datos, se debe verificar que no haya valores repetidos. Posteriormente, no se permite ingresar valores repetidos para dicha columna. El tipo BTREE genera un árbol B en memoria utilizando los datos de dicha columna. El tipo BST, genera un árbol binario de búsqueda con los datos de dicha columna. Solo se puede un índice a la vez en una sola columna de la tabla. El propósito del índice es que se tenga una estructura más liviana en memoria que permita hacer búsquedas rápidas y que indique donde está el registro completo en el archivo. Cada vez que se crea un índice, se registra en el system catalog. Los índices ya existentes, se crean en memoria cuando el servidor inicia. Es decir, si en el system catalog se indica que existe el índice x, si el servidor se reinicia, se debe volver a crear en memoria.	10
007	Para seleccionar filas de una tabla específica: SELECT * | <columns> FROM <table-name> [WHERE where-statement] [ORDER BY <column> <asc | desc>] where-statement: <column-name> compare-operator [<value>] compare-operator: >, <, =, like, not El resultado del SELECT se muestra como una tabla en la interfaz del cliente. El query processor si detecta que hay una columna especificada en el WHERE para la cual hay un índice, se utiliza el índice para buscar primero. Si no hay índice, se hace una búsqueda secuencial. Si se especifca el Order by, se utiliza Quicksort para ordenar el resultado de la consulta. Puede ser ascendente o descendente	10
008	UPDATE <table-name> SET <column-name> = <new-value> [WHERE where-statement] Actualiza las filas de la tabla que cumplan la condición del WHERE. En caso que WHERE no se especifique, se actualizan todas las filas.	10

	La actualización se realiza sobre las columnas indicadas en el SET. El query processor si detecta que hay una columna especificada en el WHERE para la cual hay un índice, se utiliza el índice para buscar primero. Si no hay índice, se hace una búsqueda secuencial. Si el SET actualiza alguna columna indizada, el índice asociado deberá actualizarse.	
009	DELETE FROM <table-name> [WHERE where-statement] Elimina las filas de la tabla que cumplan la condición del WHERE. En caso que WHERE no se especifique, se eliminan todas las filas. El query processor si detecta que hay una columna especificada en el WHERE para la cual hay un índice, se utiliza el índice para buscar primero. Si no hay índice, se hace una búsqueda secuencial. Al eliminar filas, si hay índices asociados, deben actualizarse.	5
010	INSERT INTO <table-name> VALUES(<values>,<values>...) Inserta en la tabla especificada los valores para cada columna en el orden de creación de la tabla. Valida que los tipos de dato especificados sean los correctos y actualiza los índices asociados. Las columnas DateTime se especifican en String pero internamente se parsean a DateTime	10

EJECUCIÓN DE EJEMPLO

Suponga que tiene su "módulo" de powershell listo y se llama tiny-sql-client.ps1. Para cargarlo, se abre powershell y se ejecuta:

`. .\tiny-sql-client.ps1` (nótese que el primer "." es relevante e importante)

Suponga que tiene el archivo script.tinysql en la carpeta C:\ con el siguiente contenido:

```
CREATE DATABASE Universidad;

SET DATABASE Universidad;

CREATE TABLE Estudiante (

      ID INTEGER,

      Nombre VARCHAR(30),

      PrimerApellido VARCHAR(30),

      SegundoApellido VARCHAR(30),

      FechaNacimiento DATETIME
```

```
);

INSERT INTO Estudiante (1, "Isaac", "Ramirez", "Herrera", "2000-01-01
01:02:00")

INSERT INTO Estudiante (2, "Juan", "Ramirez", "X", "2000-01-01 01:02:00")

INSERT INTO Estudiante (3, "Pedro", "Herrera", "Y", "2000-01-01 01:02:00")

CREATE INDEX Estudiante_Id ON Estudiante(ID) OF TYPE BTREE

INSERT INTO Estudiante (1, "Andrés", "Ramirez", "2000-01-01 01:02:00") //Debe
dar error dado que el índice no permite duplicados

SELECT * FROM Estudiante WHERE ID = 2;

SELECT Nombre FROM Estudiante WHERE ID = 2;

SELECT * FROM Estudiante WHERE Apellido LIKE *mire* ORDER BY Nombre DESC;

DELETE FROM Estudiante WHERE ID == 1;

UPDATE SET Nombre = "Felipe" WHERE ID == 1;
```

Para ejecutar el script, se ejecuta en la misma terminal de powershell:

```
Execute-MyQuery -QueryFile C:\script.tinysql -Port 8000 -IP 127.0.0.1
```

Se asume en este ejemplo que el server está ejecutándose en la IP local y en el puerto 8000

La forma en la que el comando Execute-MyQuery va mostrando el avance y el resultado de cada sentencia, queda a criterio de los estudiantes.

SOBRE LOS ÍNDICES

El disco es lento. Suponga que tiene una tabla sin índices de gran tamaño. La única forma de buscar un registro donde la columna A tenga el valor 1, es leer registro por registro en el disco hasta encontrar lo buscado o llegar al final.

Un índice genera una estructura árbol en memoria que mapea los valores de la columna sobre la que se creó el índice a posiciones del disco. Por ejemplo, suponga que un nodo del árbol indica que el ID 3 está en la posición 12000 del disco. Al encontrar el nodo (suponiendo que el valor buscado es 3), se abre el archivo y se desplaza directamente a la posición 12000 y se lee el registro en esa posición.

ASPECTOS OPERATIVOS

- El trabajo se realizará en **parejas**.

- El uso de Git y Github es obligatorio
- La fecha de entrega será según lo especificado en el TEC Digital. Se entrega en el TEC digital, un archivo PDF con la documentación. Los estudiantes pueden seguir trabajando en el código hasta 15 minutos antes de la cita revisión oficial.
- **Deberá entregar scripts de ejemplo que muestren casos de prueba relevantes que validen posibles escenarios de error y de éxito. Debe diseñar distintos escenarios que deberían resultar en error.**
- **Debe demostrar mediante scripts de ejemplo que el uso de índices provee mayor rapidez para las consultas**

DOCUMENTACIÓN

- La documentación deberá tener las partes estándar:
    - Portada
    - Introducción
    - Tabla de contenidos (con los títulos debidamente numerados)
    - Breve descripción del problema
    - Descripción de la solución
        - Por cada uno de los requerimientos, se deberá explicar cómo se implementó, alternativas consideradas, limitaciones, problemas encontrados y cualquier otro aspecto relevante.
    - Diseño general: diagrama de clases UML con las clases relevantes que muestren el diseño orientado a objetos y los patrones de diseño aplicados

EVALUACIÓN

- El proyecto tiene un valor de 25% de la nota del curso
- Los proyectos que no cumplan con los siguientes requisitos no serán revisados:
    - Toda la solución debe estar integrada
    - La interfaz de usuario debe estar implementada e integrada
- El código tendrá un valor total de 80%, la documentación 10% y la defensa 10%. De estas notas se calculará la *Nota Final del Proyecto*.
- Aun cuando el código y la documentación tienen sus notas por separado, se aplican las siguientes restricciones
    - **Si no se entrega documentación en formato PDF, automáticamente se obtiene una nota de 0.**
    - Si no se utiliza un manejador de código se obtiene una nota de 0.
    - Si la documentación no se entrega en la fecha indicada se obtiene una nota de 0.
    - El código debe desarrollarse en C#, si no, se obtendrá una nota de 0.
- La revisión de la documentación será realizada por parte del profesor, no durante la defensa del proyecto.

- Cada estudiante tendrá 20 minutos para exponer su trabajo al profesor y defenderlo, es responsabilidad de los estudiantes mostrar todo el trabajo realizado, por lo que se recomienda tener todo listo antes de entrar a la defensa.
- Cada grupo es responsable de llevar los equipos requeridos para la revisión, si no cuentan con estos deberán avisar al menos 2 días antes de la revisión a el profesor para coordinar el préstamo de estos.
- Durante la revisión únicamente podrán participar los miembros del grupo, asistentes, otros profesores y el coordinador del área.