CSE 344
SYSTEMS PROGRAMMING
SPRING 2021

FINAL HOMEWORK
REPORT

TURKER TERCAN
171044032

HOMEWORK'S CHALLANGE:

- Trying to simulate and SQL Server, implement both Client and Server codes with multithreaded programming.
- Server will start with reading CSV file. It will load this file into memory and work with that like an SQL Database table.
- Server and Clients will communicate with a Socket that its port given by an argument.
- Server creates pool of threads to handle communication between Server and Clients.
- Main thread connects to a client, and then it forwards that file descriptor to not busy thread.
- Server is working like a daemon process and there is no chance to instantiate it for the second time.
- Every client sends bunch of queries to the server, and the server forwards it to an available pool thread.
- Every pool thread is racing to each other. Some of them work like writers and some of them works like readers.
- Once a SIGINT signal arrives, it will stop pending queries to the pool threads and when they will be finished also main threads finishes.

DESIGN CHOICES:

Parsing CSV Table and Data Structure:

- I created another class for database (Database.c and Database.h) to read and store the CSV file.
- I used an optimized linked list for storing every row(records). In every row node, there is a string array that its size is equal to column size of the csv file.
- Read CSV file line by line and store it in the linked list.
- I used a last node in the linked list so I can add the last node a new node, instead of iterating all nodes from the beginning.
- I know getting a node from the list takes O(n) time, so optimized it in a way that, when it needs to iterate list from the beginning, I iterated first node, so it takes O(n) time instead of O(n^ 2) time.

Mutex and Condition Variables:

- I only used monitors for solving race conditions.
- I used an integer to check how many available pool threads are there. If there is no thread available in the pool. Main thread waits that condition. Otherwise, it forwards connected file descriptor to the first available thread and decrements the available pool integer, signals selected pools thread.
- I created a condition variable for each thread so, main thread can wake the selected thread up.

- Every pool thread contains two race condition problems. One of them is multithreaded server client model and the other is readers and writers problem about access to database.

- Every thread has busy flag and connection file descriptor to handle. When the server initialized, pool threads wait its own condition variable and checks busy flag and connection file descriptor is forwarded to itself.

- After the thread waken up, it parses the SQL Query. I implemented readers and writers problem like we learned in the class. Writers have more priority than the Readers. If an UPDATE SQL Query arrives, makes sures that there is no reader process access critical region. And when there is no writer query, it broadcasts to all waiting reader process.

- When there is no query to execute, closes the connection and increment available pool integer.

Client

- Client reads the given file. It only copies the lines that is spaced for its id.
- Connects to the socket and writes to socket one query.
- It reads the response and prints it.
- Also, I check, if there is no space between comma and next string, I shifted one space to the right.
- If there is no query to send, it closes the connection and terminates.

SQL Parsing

- I implemeted all the functions you asked for. SELECT, SELECT DISTINCT, UPDATE and WHERE conditions.

Tests And Valgrind:

I tried several csv files and all of them works properly. You can see the queries, csv file and bash script for the clients in the Test Case folder. I run 260 clients at the same time and all of them worked properly. Also, there is Valgrind outputs, server and clients outputs in the same folder.

Final Notes:

- Daemon process created and double instantion of the server is blocked.
- All the memory and recources are freed when server is terminated.
- SIGINT signal implemented and only way to terminate the server is to send SIGINT signal.
- All the synchronization is done by monitors.
- SELECT, SELECT DISTINC, UPDATE SQL commands are implemented.
- All of the requirements has done.