CSE-381: Systems 2

Homework #9

Due: Wed April 7 2021 before 11:59 PM

Email-based help Cutoff: 5:00 PM on Tue, April 6 2021

Maximum Points for: 34 points

Objective

The objective of this part of the homework is to develop a multithreaded database:

- Enable multithreaded `select` and `update` operations to a multithreaded database
- Enable `wait select` and `wait update` operations using condition variables
- Limit number of threads to mitigate DoS attacks
- Work with different classes and data types to build strength is problem-solving
- Understand functional testing with multithreaded programs
- **Do not use global variables**

Submission Instructions

This homework assignment must be turned-in electronically via the CODE Canvas plug-in. First ensure your program compiles without any warnings or style violations. Ensure you have tested operations of your program as indicated. Once you have tested your implementation, upload just your updated source files `SQLAir.h` and `SQLAir.cpp`.

General Note: Upload each file individually to Canvas. Do not upload archive file formats such as zip/tar/gz/7zip/rar etc.

Grading Rubric:

The programs submitted for this homework **must pass necessary base case test(s) in order to qualify for earning any score at all**. Programs that do not meet base case requirements will be assigned zero score!
Program that do not compile, have a method longer than 25 lines or badly formatted code as in } } }, etc., or some skeleton code will be assigned zero score.

- Point distribution for features:
  - Base case: Complete implementation of `selectQuery` and `updateQuery` methods to process multithreaded queries [**12 points**]
  - Additional feature #1: Extend `selectQuery and updateQuery` methods to wait-and-try (sleep-wake-up) until at least 1 row is selected or updated [**10 points**]
  - Additional feature #2: Limit number of background threads to mitigate Denial of Service (DoS) attacks using sleep-wake-up approach [**6 points**].
  - Formatting, Good organization, code reuse, documentation, etc. – **earned fully only if all of the functionality are attempted**: [2 + 2 + 2 = **6 points]**.
- **-1 Points**: for each warning generated by the compiler when using Miami University C++ project (warnings are most likely sources of errors in C++ programs)
- **-30 Points**: If global variables (without namespaces) are used.

Develop a multithreaded SQL-like database program

Background & setup

This project requires you to continue to add features to the `SQLAir` multithreaded database. The project has been developed with the following objectives:

- Combines the key topics learned in this course (and in pre-requisite courses).
  Remember: CSE graduates don't just use databases (ISA, Stats, and many other disciples also use many different databases), they build it – building systems is the hallmark of a CSE graduate that makes you unique.
- Provides a very interesting project to showcase your learning on your resumes and during job interviews.

An updated zip file is provided with the solution from previous part of this project. **You may reuse your solution from previous part**, but ensure you get the new set of test files & updated `getAndLoad` method from the zip.

An overview of the project, the processing of setting up the project, and testing are demonstrated in SQLAir Overview video on Canvas. Please review the video for the following information. The time (in `minute:second`) format where each topic starts in the video is listed below:

- Why this project – Time: `1:45` (1 minute, 45 seconds into the video)
- Demo of SQL Air – Time: `3:15`
- Setting up SQL Air project in NetBeans – Time: `6:10`
- Overview of starter code – Time: `10:18`
- Code overview in NetBeans – Time: `13:41`
- Review using debugger – Time: `15:11`
- Multithreaded testing of SQL Air – Time: `17:50`

Base case: MT-safe `selectQuery` and `updateQuery` methods [12 points]

Part #1: Complete implementation of `updateQuery` method:

1. Count and print the number of rows updated at the end of the `updateQuery` method in the format – "`5 row(s) updated.`".
2. Implement feature to handle `where` clause (*i.e.*, the `whereColIdx` parameter is not `-1`) with 3 different conditions ('=', '<>', and '`like`') in a query, as in "`update test.csv set raters=3 where title like 'The';`". See `SQLAirBase::matches` helper method. For this feature you only print rows with column-values (*i.e.*, value in column indicated by `whereColIdx`) that match the given condition. See `SQLAirBase::matches` helper method. Note that given a `row`, you can get value via `row.at(whereColIdx)`.
3. Of course, you can use the `selectQuery` method to help guide your implementation.

Sample test and output:

```
$ sql-air> select title, rating, raters from test.csv where movieid = 193579;
title   rating  raters
Jon Stewart Has Left the Building     3.5     1
1 row(s) selected.
sql-air> update test.csv set rating=3.25, raters=3 where movieid = 193579;
1 row(s) updated.
sql-air> select title, rating, raters from test.csv where movieid = 193579;
title   rating  raters
Jon Stewart Has Left the Building     3.25    3
1 row(s) selected.
```

Part #2: Make select and update queries MT-safe:

1. Lock-and-unlock each row (see `CSVRow::rowMutex`) prior to processing. Locking at the row-level is pretty standard in databases and provides a better balance (when compared to locking a whole `CSV`) between performance and thread-safety (*i.e.*, avoiding race conditions).
2. Ensure you keep your critical sections as short as possible. Do not do I/O in your critical section.
    1. For the `selectQuery` method it would be better to make a copy of the row (inside your critical section) you are operating on and use the copy for I/O outside the critical section.
3. To further minimize unnecessary I/O, **modify the `selectQuery`** to print column titles only if at least 1-row is selected.

Automated multithreading test:

Use the supplied `mt_tester` program to do multithreading testing as follows:

1. First run the your SQL Air serer on a given port number by specifying the port number as a command-line argument (*e.g.*, `./sqlair 8080`). **Note**: For each round of testing below, you will need to stop and restart your server as the data is updated/changed in the second test (*i.e.*, `mt_sel_upd_tests.txt`).
2. Next in a separate terminal run the `mt_tester` program 2 separate times as shown below:

```
$ ./mt_tester tests/mt_select_tests.txt 8080
Finished block #0 testing phase.
Finished block #1 testing phase.
Testing completed.
```

```
$ ./mt_tester tests/mt_sel_upd_tests.txt 8080
Finished block #0 testing phase.
Finished block #1 testing phase.
Testing completed.
```

Additional feature #1: Wait-queries using sleep-wake up approach [10 points]

Background: Wait-queries are unique to SQLAir. In these queries, the user can request the database to wait and retry `select` or `update` operations until at least one row is selected or updated. The wait-queries are specified by adding the wait clause to a select or update statements as in:

```
sql-air> wait select * from test.csv where raters = 2;
sql-air> wait update test.csv set rating=5 where raters = 3;
```

Feature requirements: The `selectQuery` and `updateQuery` methods accept a `mustWait` flag. This flag is set to true when the user specifies a `wait` clause. Using the value of the `mustWait` flag, the two methods must operate in the following manner:

1. If the `mustWait` flag is `false`, then the methods must continue to operate the same manner as in the previous part(s).
2. Else, if the `mustWait` flag is `true`, then the methods must try to select/update a CSV until at least 1 row is selected/updated, while honoring any conditions specified via a `where` clause. If no rows were selected/updated then the methods must wait and retry the operation after an update has occurred to the specific `CSV` (indicating some data has changed and the queries may succeed).
    - For the sleep-wake up approach use `CSV::csvMutex` and `CSV::csvCondVar` (already in the starter code).
    - For this project, use an unconditional wait – *i.e.*, `csvCondVar.wait(lock)` in both `select` and `update` options (or their helper methods based on your solution).
    - Since we don't know which conditions change, use `notify_all()` to wake-up all awaiting threads after an `update` or `wait update` statement updates at least 1 row.

Manual Testing:

For initial testing, it would be easiest to use two separate tabs in a web-browser as demonstrated in the adjacent video:

1. In the first tab run a `wait select` or `wait update` statement.
2. After that, in second tab run an `update` statement that changes the data.
3. Once the data is changed, the query in the first tab should show updated results.

Despite all the testing, it still takes a human to review a program to decide if a program is correctly multithreaded. Hence, the even if your program passes all the tests in the CODE plug-in it could still be incorrectly multithreaded. Consequently, double-check your solution to ensure you are correctly multithreading your program in order to earn full points for this homework.

Automated testing:

Use the supplied `mt_tester` program to do multithreading testing as follows:

1. First run the your SQL Air serer on a given port number by specifying the port number as a command-line argument (*e.g.*, `./sqlair 8080`). **Note**: For each round of testing below, you will need to stop and restart your after each test.
2. Next in a separate terminal run the `mt_tester` program as shown below:

```
$ ./mt_tester tests/mt_wait_sel_tests.txt 8080
Finished block #0 testing phase.
Finished block #1 testing phase.
Finished block #2 testing phase.
Finished block #3 testing phase.
Finished block #4 testing phase.
Finished block #5 testing phase.
```

```
Finished block #6 testing phase.
Finished block #7 testing phase.
Testing completed.
```

3. Stop and restart your server and run the `mt_tester` program as shown below:

```
$ ./mt_tester tests/mt_sel_upd_tests.txt 8080
Finished block #0 testing phase.
Finished block #1 testing phase.
Finished block #2 testing phase.
Finished block #3 testing phase.
Finished block #4 testing phase.
Finished block #5 testing phase.
Testing completed.
```

Additional feature #2: Limit number of background threads [6 points]

In the previous multithreading features the maximum number of detached threads were not limited (to keep the problem simple). Unlimited threading leaves the server highly susceptible to Denial-of-Service (DOS) attacks, which is a big cybersecurity issue. Hence, in this feature you are required to limit the maximum number of detached threads to be fewer than the `maxThr` parameter to the `runServer` method using a Sleep-Wake-up approach. **The solution must use a Sleep-Wake-up approach to earn any points for this feature**.

Implementation suggestions

There are already a couple of variables and tips (see documentation for the variables) to implement this feature:

1. See atomic variable `numThreads` in `SQLAir.h` header. Increment and decrement this variable appropriately when a thread is started/stops.
2. See `thrCond` condition variable (in `SQLAir.h`). You can use this condition variable to wait until the number of detached threads is less than `maxThr`.

Manual Testing

There isn't a good approach to automatically test this feature. You have to manually test this feature via:

1. Limit the number of threads to 2 and run your server:

```
$ ./sqlair 8080 2
```

2. Next, use 2 separate tabs on a web-browser to run `wait select` queries that will wait forever. These two queries will now hold-up the 2 background threads.

3. Now, open a 3rd tab and try to access the server. You should notice the web-page does not load (because we have limited to 2 threads and the server has run out of threads to service the 3rd request).

Note: This feature is not automatically testable and requires manual code review. Hence, on Canvas, the results from this test may appear to fail and that is ok.

Submit to Canvas

This homework assignment must be turned-in electronically via CODE Canvas plug-in. Ensure all of your program(s) compile without any warnings or style violations and operate correctly. Once you have tested your implementation, upload:

1. Your updated source files `SQLAir.h` and `SQLAir.cpp`,

Upload each file associated with homework (or lab exercises) individually to Canvas. Do not upload archive file formats such as zip/tar/gz/7zip/rar etc.

Despite all the testing, it still takes a human to review a program to decide if a program is correctly multithreaded. Hence, the even if your program passes all the tests in the CODE plug-in it could still be incorrectly multithreaded. Consequently, double-check your solution to ensure you are correctly multithreading your program in order to earn full points for this homework.