CMSC 624 Database System Architecture and Implementation **Spring 2025**

Programming Assignment 2

Assigned: February 24 *Due: March 10, 11:59:59 PM*

1 Description

In this assignment you will be implementing six concurrency control schemes:

- Two versions of locking schemes, both of which implement variations of the standard two-phase locking algorithm we discussed in class.
- A version of OCC very similar to the serial-validation version described in the OCC paper you read for class.
- A version of OCC somewhat similar to the parallel-validation version described in the OCC paper.
- A version of MVCC timestamp ordering that is a simplified version of the PostgreSQL scheme we studied in class.
- A serializable version of the MVCC scheme that is a simplified version of PostgreSQL's SSI scheme.

2 Requirements

1. Your code must be submitted as a series of commits that are pushed to the origin/master branch of your private Git repository. We consider your latest commit prior to the due date/time to represent your submission.
2. The directory for your project must be located at the root of your Git repository.
3. If your code is not compiled and tested on the test server, you will receive 0 points for the implementation.
4. You are not allowed to work in teams or to copy code from any source.
5. You are not allowed to discuss your analysis with anyone else.
6. Please submit ({your_name}_a2_sol.pdf) to the assignment 2 link on ELMS. You do not need to submit any code, since we can access to your repositories.

3 Part 5

3.1 Carpe datum (0 points)

When you finish the coding part of this assignment, we will run your code on our test server, and commit the results back to you.

3.2 Simulations are doomed to succeed. (4 points)

Transaction durations are accomplished simply by forcing the thread executing each transaction to run in a busy loop for approximately the amount of time specified. This is supposed to simulate transaction logic — e.g. the application might run some proprietary customer scoring function to estimate the total value of the customer after reading in the customer record from the database. Please list **at least two weaknesses** of this simulation — i.e. give two reasons why performance of the different concurrency control schemes you experimented with for this assignment would change relative to each other if we ran actual application code instead of just simulating it with a busy loop.

`a: - your answer here...`

3.3 Locking manager (6 points)

1. Is deadlock possible in your locking implementation? Why or why not?
   `a: - your answer here...`

2. Most 2PL systems wait until they access data before they request a lock. But this implementation requests all locks before executing any transaction code. What is a performance-related disadvantage of our implementation in this assignment of requesting all locks in advance? What is a client-usability disadvantage of requesting all locks in advance?
   `a: - your answer here...`

3.4 OCCam's Razor (6 points)

The OCC with serial validation is simpler than OCC with parallel validation.

1. How did the two algorithms compare with each other in this simulation? Why do you think that is the case?
   `a: - your answer here...`

2. How does this compare to the OCC paper that we read for class?
   `a: - your answer here...`

3. What is the biggest reason for the difference between your results and the what you expected after reading the OCC paper?
   `a: - your answer here...`

If you did not follow the given pseudocode for OCC with parallel validation, give your pseudocode and argue why it is better.

3.5 OCC vs. Locking B (7 points)

If your code is correct, you probably found that relative performance of OCC and Locking B were different from the tradeoffs we discussed in class. In fact, you might be quite surprised by your results. Please describe the two biggest differences between the relative performance of OCC vs. Locking B relative to what we discussed in class, and explain why the theory doesn't match the practice for this codebase for each of these two surprises.

3.6 MVCC vs. OCC/Locking (6 points)

1. For the read-write tests, MVCC performs worse than OCC and Locking. Why?
   `a: - your answer here...`

2. MVCC even sometimes does worse than serial. Why?
   `a: - your answer here...`

3. Yet for the mixed read-only/read-write experiment it performs the best, even though it wasn't the best for either read-only nor read-write. Why?
   `a: - your answer here...`

If you wrote your own version, please explain why it's better than the ones presented here.

3.7 MVCC pseudocode (4 points)

1. Why did our MVCC pseudocode request read locks before each read?
   `a: - your answer here...`

2. In particular, what would happen if you didn't acquire these read locks?
   `a: - your answer here...`

3. How long do these locks have to be held?
   `a: - your answer here...`

4. Does the MVCC pseudocode guarantee serializability? (we are asking specifically about the pseudocode from this assignment — not about MVCC in general or the version from PostreSQL we discussed in class). Please provide a brief explanation of why or why not (if not, please provide an example).
   `a: - your answer here...`

3.8 Mixed transaction lengths (8 points)

Take a close look at your high contention read/write results. When transaction lengths are uniform, one of the conccurency control schemes you implemented is best. However, for mixed transaction lengths (the fourth column in the results), you will probably see a different concurrency control scheme as the winner! Please explain why the results changed for mixed transaction lengths.

`a: - your answer here...`