Database Course Assignment 2: MVCC and Garbage Collection

April 15, 2025

1 Introduction

Building on the previous assignment, where you implemented an ART with versioning using the O2N (oldest-to-newest) approach, this assignment extends your work in several directions. You will implement Snapshot Isolation MVCC and Serializable MV-OCC in a multi-threaded environment, and design garbage collection (GC) mechanisms for the version chain.

2 Assignment Overview

In this assignment, you are required to:

1. Implement Snapshot Isolation MVCC.
2. Add Serializable check by OCC.
3. Make sure your code is thread-safe.
4. Implement garbage collection (GC) for version chain. For O2N version link, although the approach seems straightforward (traverse the chain, check for expired versions, and delete them), proper concurrency control is required. A naive solution is to lock the entire version chain during GC. While correct, this approach may significantly impact performance.
5. (Optional Bonus) Propose and implement alternative GC schemes or discuss advanced MVCC optimizations (e.g., dynamic timestamp assignment).

3 Simplifying Assumption

In this assignment, all tests follow the same method, that is, first insert some tuples by a single thread, then do a lot of update operation by single-thread or multi-thread. All tables are initialized with an ART index, and every update operation will not change the key. So, the index is almost static, i.e. no new key will be inserted after the initializing stage.

4 Task 0: Merge with the updates in the codebase

To support Concurrency Control, some interface of ART Index is updated. Make minor modifications to your code to adapt to the new interface.

5 Task 1: Snapshot Isolation MVCC

- Implement Snapshot Isolation MVCC.
- Design your solution to operate correctly in a multi-threaded environment.
- Recall MVCC: Each transaction will be given a read timestamp when created. When the transaction reads the database, it will use the timestamp to fetch the data.
- In this assignment, each row can only be updated by at most **1** uncommitted transaction, which means that the version link can contain at most one uncommitted version node.

- Read-only transactions should always be able to be committed. In this part, you should only detect write-write conflicts.

6 Task 2: Serializable MV-OCC

- Implement Serializable MV-OCC.
- Design your solution to operate correctly in a multi-threaded environment.
- Recall OCC: Before commit, the transaction, except for the read-only transaction, will check if every row it has read has changed since it started. If so, it will abort itself.

7 Task 3: Garbage Collection for O2N Version Chains

- Implement GC for O2N version chains by traversing each version chain to identify expired versions and deleting them.
- Although the GC algorithm appears simple, it must incorporate concurrency control. A naive approach is to lock the entire version chain during GC, ensuring correctness but potentially causing severe performance degradation.
- When creating or deleting new version nodes, you should call the trigger function we provide for grading.

8 Codebase Details & Grading

- For more codebase or testcases details, see the README.md in the Github repo.
- There are 8 tests with totaling 16 points.
- **Code and Report Readability:** 4 points for clear code and a well-structured report. The report should contain detailed discussions on each design.
- **Bonus points:** Bonus up to 5 points for discussion and implementation of alternative GC schemes or advanced MVCC optimizations.

Test cases will simulate the concurrent workload in a single thread and check your response or execute in a multi-thread environment directly and check some constraints.

9 Submission Requirements

- Submit the complete source code.
- Submit a detailed report that explains your design decisions, implementation details, and any optimizations.