Homework 2. CC Simulator

Due date: Dec. 4 (Thu) 23:59:59

Overview

In this project, your task is to implement database concurrency control simulator using C++.
This project includes two major concurrency control scheme: rigorous two-phase locking (R2PL) and optimistic concurrency control (OCC).

Instruction

Our concurrency control simulator reads transaction schedules one by one and processes read, write, and commit operations. For example, R1(A) represents a read request for data A by Transaction 1. Detailed transaction schedules can be found in the **tests** directory.

R2PL releases all locks (i.e., shared and exclusive lock) when a transaction commits, while OCC performs validation before committing and only writes to the main database file if the validation passes.

To simplify the implementation, neither concurrency control technique will perform actual read/write operations (e.g., modifying data directly).
You will need to modify the src/lock.cpp and src/occ.cpp files, writing your code in the sections marked "DIY." Descriptions for transactions and object structures can be found in the **include/system.h** file.

We will evaluate the answer by comparing the output files. Please do not modify print statement in the source code files. Make sure that compare your output file and solution file with diff command.

Notes

- Please read the source code carefully.
- Please do not modify other source code files except for lock.cpp and occ.cpp
- Please ensure that you compare your output with the corresponding solution file.

1. R2PL

- Implement below functions in the **src/lock.cpp** file.
- Read the description for each function and write your code at the section marked "DIY".

Function Name	Description
`bool lock::acquire_lock( trx_t, object_t, LOCK_TYPE)`	Acquire lock to the given object with lock type following these steps. step1. Find the existing locks on the object and traverse all locks step2. Check whether the transaction can acuire the requested lock. If the object is already locked by a younger transaction and there is a conflict, we will abort (rollback) the younger transaction. step3. If the transaction can acquire lock, it adds lock list

	step4. If the transaction can not acuqire lock then we need to call rollback function.
`void lock::release_lock( trx_t)`	Traverse the lock_list and remove all locks acquired by given transaction.
`void lock::rollback(trx_t)`	Rollback the transaction. step1. Release all locks held by the transaction. step2. Remove the actions performed by the transaction from the `output` vector. setp3. Remove the remaining actions of the transaction being rolled back from the `actions` vector, and then append the actions of the rollbacked transaction to the end of the actions vector. step4. Reset the timestamp (YOU DO NOT NEED TO MODIFY)
`STATUS lock::execute(std::string, trx_t, object_t)`	Process the given action. A read operation must acquire LOCK_TYPE::SHARED, while a write operation must acquire LOCK_TYPE::EXCLUSIVE. Neither operation performs the actual read or write; they only acquire the lock. In this case, if the lock is successfully obtained, it returns STATUS::SUCCESS; if not, it returns STATUS::BLOCKED. If it returns STATUS::SUCCESS You have to add action into `output` vector (see COMMIT case). Upon commit, all locks held by the transaction are released, and the action is added to the output vector. Then, STATUS::COMMIT is returned.
`void lock::run()`	Perform the actions sequentially. The actions vector is parsed from the transaction schedule.

2. OCC

- Implement below functions in the **src/occ.cpp** file.
- Read the description for each function and write your code at the section marked “DIY”.

Function Name	**Description**
void occ::trx_read(trx_t&, object_t)	Read data item and records it into transation's private read set
void occ::trx_write(trx_t&, object_t)	Write data item and records it into transaction's private write set. This function does NOT write to the main database file.
bool occ::trx_validate(trx_t&)	Validate the transaction. When transaction try to commit operation; we first call this function (see the `run()` function.)
void occ::commit(trx_t&)	Commit the transaction.

3. How to compile

```
cd [repository-name]
mkdir bld && cd bld
cmake ..
make -j
```

4. Run and Test

```
cd [repository-name]
cd bld

# R2PL test
./lock ../tests/test1.txt &> test1.txt
diff test1.txt ../solutions/lock-sol1.out

# OCC test
./occ ../tests/test1.txt
diff test1.txt ../solutions/occ-sol1.out
```

Submission

- Due date: 2025.12.4 (Thu) 23:59
- **Do not commit or make any changes after the assignment deadline.**
- **DO NOT LEAVE THE GITHUB CLASSROOM**

How to submit

```
# You can commit and push as you can before the deadline
# For final commit message for final submission, please set the commit
message as submission-student-id (e.g., submission-123123)

git add.
git commit -m “submission-student-id”
git push
```

Late submission policy

- 75%: 1 day late
- 50%: 2 days late
- 25%: 3 days late
- 0%: 4 days and more

Warning

- Do not use ChatGPT
- Submit your homework Github Classroom only.
- Do not need to submit any assignments on KU LMS.
- Do not upload the project to public including source code files and documents
- Do not copy other student's answer
- Do not collaborate other students. This is an individual project (No groups)
- Do not modify the database file (i.e., Do not insert/delete/update in the database arbitrarily)
- For your query, the order of output columns (attribute) is very important. Please follow the instruction of the problem carefully.
- Again, we will evaluate the answer by comparing the output files. Please make sure to always verify your program works properly. (No partial points)
- Do not change your Github ID used in homework 0.