Q1
10 Points

This quiz is meant to take only 20 minutes or so. While you have over 24 hours to finish at your convenience, don't feel that you need to write a lot of text. A paragraph or two for each question could be sufficient. Try to answer in a way that demonstrates your understanding of lab 2 and 3.

Q1.1 Lab 2: Operator Algorithms
5 Points

Which implementation did you choose for your join operator in lab 2?

Describe a different implementation that you think might improve performance for a particular case.

I have used two different implementations for the join operator in lab2, the first one is the nested loop join, which will handle any predicates excluding the 'equal Join'. Because I found the nested loop join is so slow for the optional challenge queries.

So the second implementation is a hash join implementation which only handles the 'equal join' predicate. It increased the speed 500 times on my computer when I run the third query in the optional challenge. However, it can only handle the predicate 'equal', so the different implementation I was thinking about implementing is the sort merge join. Because it might be able to take care of more predicates, such as 'less equal join', 'greater equal join' and so on.

In summary I have implemented nested loop join, hash join implementation in lab2. And the sort merge join might improve the performance for 'less equal join', 'greater equal join', which hash join implementation can not handle by itself.

Q1.2 Lab 3: Transactions
5 Points

Briefly explain how you implemented shared and exclusive locking in your implementation. What happens when a transaction requests an exclusive lock on an element that has multiple shared locks already?

First of all I chose to set my locking granularity on page-level for both shared and exclusive lock.

When we have a new transaction, it will grab a sharedLock when the transaction is calling getPage() from the buffer pool, and the sharedLock will be released when the transaction commits or aborts.

When the transaction is calling the writePage(), it will get an exclusive lock on the page that is writing, so when the lock manager receives the requests of exclusive lock on an element from the certain transaction, My lock Manager will first check is there any exclusive lock on this page, if not it will check how many transactions other than the one send the request are holding the shared locks on the page.

If the element has already got multiple shared locks already, the first thing my lock manager will do is to check if there are any deadlock, if there are no deadlocks. The transaction that requesting the exclusive lock on the same element(Page) will need to wait for a period of time and let other transactions finish their jobs, and after the wait, it will try to grab an exclusive lock again. If there are still multiple transactions having the shared lock on that page, it will wait again, Finally, it will find no other transactions has the shared lock on the same page at some point. When it finally happens it will not hold the shared lock on the page, but instead hold a new exclusive lock!

Quiz 2+3 — GRADED

STUDENT
Yuanfeng Li

TOTAL POINTS
10 / 10 pts

QUESTION 1
(no title) **10** / 10 pts
1.1 Lab 2: Operator Algorithms **5** / 5 pts
1.2 Lab 3: Transactions **5** / 5 pts

Select a question.

Submission History | Request Regrade | Next Question >