COMP90050 Advanced Database Systems: Tutorial
Winter term, 2023 (Week 3) - solution

Exercises

Part 1

1. Discuss why the isolation property of ACID properties will apply to both an Online shopping platform as well as an Online banking system despite that they are different applications dealing with different data.

   Solution:

   Both types of systems may need to serve a large number of customers at any given time. For achieving consistency, both systems require that a transaction does not use the values that have been modified by another uncommitted transaction. A naïve approach to achieve this is handling one customer at a time. But the efficiency of the service would be extremely low with this approach. Therefore, the systems should allow different transactions to run concurrently. At the same time, the changes to the data are made as if the transactions run on a serial schedule, i.e., a transaction runs as if it is the only transaction in the system and does not become aware of other concurrent transactions which is the objective of isolation. Isolation helps with achieving a high level of efficiency while maintaining the consistency of the data that is manipulated.

2. A bank with millions of customers provides a bonus to each of its customer at the end of the year. The bonus is updated in the database as a flat transaction shown below. Discuss example and the associated issue(s) that can happen with such execution. Is it a good choice to use flat transaction here?

```
GiveEndofYearBonus()
{
        exec sql BEGIN WORK
        for each customer in database
         {
           double bonus = calculate_bonus(customer);
           exec SQL UPDATE customer
              set account = account + :bonus
         }
        exec sql COMMIT WORK
}
```

Solution:

The customer table here is a large table with millions of customers. This means this transaction can potentially be doing a lot of work during its execution and take a long time. Any failure in this transaction's execution would mean a lot of work lost. So better not to have this transaction as a flat transaction.

3. A flat transaction with save-points has the following statements as example. If condition1 is true once and the final commit is successful, what will be printed as the value of count?

```
BEGIN WORK
count = 10
SAVE WORK 1
count = count+10
SAVE WORK 2
count = count+5
SAVE WORK3
count = count+5
If (condition1) ROLLBACK WORK(2)
count = count+1
print count
COMMIT WORK
```

If condition1 is true, then the transaction rolls back to the save point WORK2. The execution order then follow the order below, with the value of count shown in the sides. The final value of count is 21

```
BEGIN WORK
count = 10
SAVE WORK 1
count = count+10  //count is 20 at this point
SAVE WORK 2
~~count = count+5~~
~~SAVE WORK3~~
~~count = count+5~~
~~If (condition1) ROLLBACK WORK(2)~~
count = count+1 // count is 21 at this point
print count //21 will be printed as the value of count
COMMIT WORK
```

4. In a nested transaction, a transaction PARENT has three sub-transactions A, B, C. For each of the following scenarios, answer which of these four transactions' commits can be made durable, and which ones has to be forced to rollback.

(i) Scenario 1: Commit by A, B, and C; but PARENT rolls back. – based on the rollback rule, all four transactions rolls back, no durable commit by any transaction.

(ii) Scenario 2: Commit by A, B, C, and PARENT. – all four transactions make durable commits, no roll back.

(iii) Scenario 3: Commit by A, B, and PARENT; but C rolls back. - Durable commits by A,B, and Parent; roll back by C only.

Part 2

5. What is the probability that a deadlock situation occurs?

It can also be found in Section 7.11.5 of the reference book - Transaction Processing, Jim Gray and Andreas Reuter, Morgan Kaufmann, 1992.

Assume,

Number of transactions = $n+1 \approx n$ (if n is large)

Each transaction access r number of locks exclusively

Total number of records in the database = R

On average, each transaction is holding r/2 locks approximately (minimum 0 and maximum r, hence average is r/2) – transaction that just commenced holds 0 locks, transaction that is just about to finish holds r locks.

Average number of locks taken by the other n transactions = n*(r/2) =

The probability that a particular transaction waits for a lock = Requesting an exclusive lock on one of the nr/2 locks held by the other n Transactions out of R possible locks = $\frac{nr}{2}$ out of potential R records = $\frac{nr}{2R}$

The probability that a particular transaction waits for a lock = (nr)/2R
The probability that a particular transaction did not wait for a lock = 1- (nr)/2R
The probability that a particular transaction did not wait (for any of the r locks), P = (1- (nr)/2R)^r
$\approx 1- (nr^2/2R)$
[note that $(1+\varepsilon)^r = 1+r*\varepsilon$ when ε is small]
The probability that a transaction waits in its life time pw(T) = 1-P = 1 – {1- $(nr^2/2R)$} = $nr^2/2R$
Probability that a particular Transaction T waits for some transaction T1 and T1 waits for T is
= pw(T)* (pw(T1)/n) = $nr^4/4R^2$ [since the probability T1 waits for T is 1/n times the probability T1 waits for someone]
Probability of any two transactions causing deadlock is = n* $nr^4/4R^2$ = $n^2r^4/4R^2$
(This is a very small probability in practice)

6. If we use the following comments to lock and unlock access to objects, then which transactions below are in deadlock if they start around the same time?

T1	T2	T3	T4
Begin	Begin	Begin	Begin
LOCK(C)	LOCK(A)	LOCK(C)	LOCK(B)
Write C	Write A	Write C	Write B
UNLOCK(C)	LOCK(B)	UNLOCK(C)	LOCK(A)
End	Write(B)	End	Write A
	UNLOCK(A)		UNLOCK(A)
	UNLOCK(B)		UNLOCK(B)
	End		End

Solution:
T2 and T4 are in a deadlock as each of them will wait for the other to release a lock while holding a lock that the other needs to acquire to complete.

7. Isolation property in ACID properties states that each transaction should run without being aware or in interference with another transaction in the system. If that is the case, we can run transactions sequentially by locking the whole database itself and adhering to the Isolation property through a big lock per transaction. Review why this may not be an ideal solution.

   Solution:

   This is in fact one type of, simplistic, concurrency control, i.e., making sure that all transactions run in a sequence, i.e., in some order. But then we are not benefiting from the potential use of idle resources such as accessing disk while another transaction is using the CPU. So even under single CPU situations, concurrency can speed up things that we are not doing. In addition, if there is a long-running transaction in the system, holding up every other transaction till that long one finish is going to make the associated company's customers very unhappy. So in short, although we want our executions to be equal to the outcome of a serial execution of a given set of transactions, we do not really want to run them one after the other but rather in concurrency with each other.

8. Given two transactions, per operation of each transaction, we can use locks to make sure concurrent access is done properly to individual objects that are used in both transactions i.e., they are not accessed at the same time. This is after all what the operating systems do, e.g., lock a file while one program is accessing it so others cannot change it at the same time. Give two transactions showing that this is not enough to achieve the isolation property of transactions for RDBMSs.

Solution:

Answer (A is initially 0 on disk)	Transaction 1 at Process 1	Transaction 2 at Process 2
Operation 1	Lock(A)	
Operation 2	Read(A)	
Operation 3	Unlock(A)	
Operation 4	A = A +100	
Operation 5		Lock(A)
Operation 6		Read(A)
Operation 7		A = A + 50
Operation 8		Write(A)
Operation 9		Unlock(A)/Commit
Operation 10	Lock(A)	
Operation 11	Write(A)	
Operation 12	Unlock(A)/Commit	

This execution order above is not equal to T1 running first nor to T2 running first and does not obey the Isolation property, although, for each disk access, we first obtained a lock properly. Thus we need something more for proper concurrency control in DBMSs.

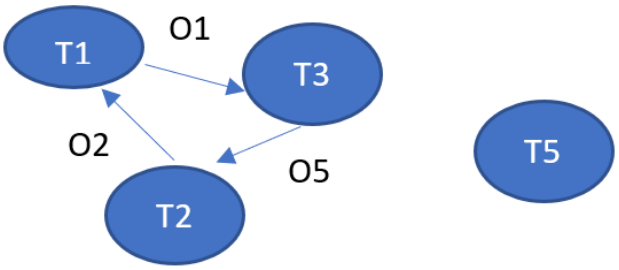
Part 3

9. What are the dependencies in the following history (a sequence of tuples in the form (T i, Oi, T j))? Draw the dependency graph mapping to this dependency set as well.

   H =< (T1,R,O1),(T3,W,O5),(T3,W,O1),(T2,R,O5),(T2,W,O2),(T5,R,O4),(T1,R,O2),(T5,R,O3) >
   Solution:

   Dependencies are given as:

   DEP(H) = < T1,O1,T3 >,< T3,O5,T2 >,< T2,O2,T1 > We can also build the following dependency graph based on this like the following:



10. Given the solution above for the previous question, can we say the history is equal to a serial history? If yes, show one such history. If not, show that there is a wormhole.

    Solution:
    There exists a cycle in the dependency graph, i.e., there are wormhole transactions (e.g., T1 is before and after of T3 at the same time). Therefore, finding equivalent serial execution is not possible.

11. Assume the following two transactions start at nearly the same time and there is no other concurrent transaction. The 2nd operation of both transactions is Xlock(B). Is there a potential problem if Transaction 1 performs the operation first? What if Transaction 2 performs the operation first?

Transaction 1		Transaction 2	
1.1	Slock(A)	2.1	Slock(A)
1.2	Xlock(B)	2.2	Xlock(B)
1.3	Read(A)	2.3	Write(B)
1.4	Read(B)	2.4	Unlock(B)
1.5	Write(B)	2.5	Read(A)
1.6	Unlock(A)	2.6	Xlock(B)
1.7	Unlock(B)	2.7	Write(B)
		2.8	Unlock(A)
		2.9	Unlock(B)

:

If Transaction 1 executes Step 1.2 before Transaction 2 executes Step 2.2, there would be no problem. This is because T1 releases the lock at the end. T2 has to wait till then. However, if Transaction 2 executes Step 2.2 first, Transaction 1 may have a dirty read of B if it tries to run Step 1.2 immediately after Transaction 2 releases the Xlock on B. This is because when Transaction 2 releases the lock on B (Step 2.4), Transaction 1 will be granted the Xlock on B (Step 1.2). After that, Transaction 1 will read B (Step 1.4). After Transaction 1 completes, Transaction 2 will acquire another Xlock on B (Step 2.6) then modify the object. In other words, the reading of B by Transaction 1 happens between two writes of B by Transaction 2.

12. What degree of isolation does the following transaction provide?

Slock(A)
Xlock(B)
Read(A)
Write(B)
Read(C)
Unlock(A)
Unlock(B)

Solution:
Degree 1 as there is one read operation 'Read(C)' without taking any lock. The only write operation has an exclusive lock associated with it. The transaction is two-phase with respect to exclusive lock.

13. The following operations are given with Degree 2 isolation locking principles in place. Convert the locking sequence to Degree 3.

Degree 2
Slock(A)
Read(A)
Unlock(A)
Xlock(C)
Xlock(B)
Write(B)
Slock(A)
Read(A)
Unlock(A)
Write(C)
Unlock(B)
Unlock(C)

Solution

Degree 3
Slock(A)
Read(A)
Xlock(C)
Xlock(B)
Write(B)
Read(A)
Unlock(A)
Write(C)
Unlock(B)
Unlock(C)