TDT4145: Data Modelling and Database Systems

Exercise 5: “Transaction management, logging and normalization”

David Rozas Domingo
Miguel Bono Tur

TASK 1

a) Explain the ACID properties.

ACID=atomicity, consistency, isolation and durability.

1. Atomicity refers to the ability of the DBMS to guarantee that either all the tasks of a transaction are performed or none of them are. Users should not have to worry about incomplete transactions.
2. Consistency property ensures that the database remains in a consistent state before the start of the transaction and after the transaction is over (whether successful or not). Ensuring this property of a transaction is the responsibility of the user.
3. Isolation refers to the ability of the application to make operations in a transaction appear isolated from all other operations. This means that no operation outside the transaction can ever see the data in an intermediate state.
4. Durability refers to the guarantee that once the user has been notified of success, the transaction will persist, and not be undone. This means it will survive system failure, and that the database system has checked the integrity constraints and won't need to abort the transaction.

b) Explain what a protocol using two-phase locking is

Also called *Strict 2PL*. It has two rules:

1. If a transaction T wants to read (respectively, modify) an object, it first requests a shared (respectively, exclusive) loc on the object.
2. All locks held by a transaction are released when transaction is completed.

Requests to acquire and release locks can be automatically inserted into transactions by the DBMS.

The locking protocol allows only 'safe' interleavings of transactions. If two transactions access completely independent parts of the database, they concurrently obtain the locks they need and proceed on their ways. If two transactions access the same object, and one wants to modify it, their actions are effectively ordered serially.

An example:

$$D = \begin{bmatrix} T1 & T2 \\ S(A) & \\ R(A) & \\ & S(A) \\ & R(A) \\ & X(B) \\ & R(B) \\ & W(B) \\ & Commit \\ X(C) & \\ R(C) & \\ W(C) & \\ Commit & \end{bmatrix}$$

where:

- S(O) is a shared lock action on an object O
- X(O) is an exclusive lock action on an object O
- R(O) is a read action on an object O
- W(O) is a write action on an object O

TASK 2

a) **Draw the precedence graphs for the schedules and decide which schedules are conflict-serializable.**

S1: w2(X); w1(X); r3(X); r1(X); w2(Y); c1; r3(Y); r3(Z), c3; r2(X); c2;

T1	**T2**	**T3**
	W(X)	
W(X)		
		R(X)
R(X)		
	W(Y)	
Commit		
		R(Y)
		R(Z)
		commit
	R(X)	
	commit	

Serializable: for example T3 can do R(X) before that T1 do R(X).
If T1 is aborted T2 (R(X)) and T3 (R(X)) can be aborted too. But if T2 is aborted T3 (R(Y)) are commited, so is unrecoverable. We need a **recoverable schedule**.

S2: r3(Z); r3(Y), w2(Y); r2(Z); w2(X); w1(X); c2; r1(X); c1; r3(X); c3;

T1	**T2**	**T3**
		R(Z)
		R(Y)
	W(Y)	
	R(Z)	
	W(X)	
W(X)		
	commit	
R(X)		
commit		
		R(X)
		commit

Serializable: for example T3 can do R(Z) after T2 do it commit.
There is no problem with T2. If T1 is aborted T3 (R(X)) is not committed, so **cascading** is good.

S3: r3(Z); w2(X); w2(Y); r1(X); r3(X); r2(Z), c2; r3(Y), c3; w1(X); c1;

T1	T2	T3
		R(Z)
	W(X)	
	W(Y)	
R(X)		
		R(X)
	R(Z)	
	commit	
		R(Y)
		commit
W(X)		
commit		

Serializable: for example T3 can do R(Z) after T2 do it commit.

If T2 is aborted, T1 (R(X)) and T3 (R(Y)) are not committed, so **cascading** is ok.

S4: r2(Z); w2(X); w2(Y), c2; w1(X); r1(X); c1; r3(X); r3(Z); r3(Y); c3;

T1	T2	T3
	R(Z)	
	W(X)	
	W(Y)	
	commit	
W(X)		
R(X)		
commit		
		R(X)
		R(Z)
		R(Y)
		commit

Serializable: for example T3 can do R(Z) before T2 start.

Cascading is ok because if T1 is aborted T3 (R(X)) are not commited. The between T2 and T3 (R(Y)).

S5: r1(X); r2(X); w2(X); w2(Y); c2; w1(X); r3(Z); w1(Y); c1; r3(Y); r3(X); c3;

T1	**T2**	**T3**
R(X)		
	R(X)	
	W(X)	
	commit	
W(X)		
		R(Z)
W(Y)		
Commit		
		R(Y)
		R(X)
		commit

Serializable: for example T2 can do R(X) before T1 start.

Cascading is ok because if T1 is aborted T3 (R(X) and R(Y)) are not commited.

S6: r2(X); w2(X), r1(X); r2(Y); w1(Y); c1; r2(Z); w2(Z); c2;

T1	**T2**
	R(X)
	W(X)
R(X)	
	R(Y)
W(Y)	
commit	
	R(Z)
	W(Z)
	commit

No serializable:if we change the order of the actions the result can be different.

None recovery properties.

TASK 3

Exercise 16.6 Answer the following questions: SQL supports four isolation-levels and two access-modes, for a total of eight combinations of isolation-level and access-mode. Each combination implicitly defines a class of transactions; the following questions refer to these eight classes:

1. **Consider the four SQL isolation levels. Describe which of the phenomena can occur at each of these isolation levels: dirty read, unrepeatable read, phantom problem.**
2. **For each of the four isolation levels, give examples of transactions that could be run safely at that level.**
3. **Why does the access mode of a transaction matter?**

1.-

Level	**Dirty Read**	**Unrepeatable Read**	**Phantom**
Read Uncommitted	Maybe	Maybe	Maybe
Read Committed	No	Maybe	Maybe
Repeatable Read	No	No	Maybe
Serializable	No	No	No

2.-

Read uncommitted: R1(A); W1(A); R2(B); W2(B)

Read committed: R1(A); W2(A); R2(B); W2(B); R1(B); W1(B)

Repeatable read: R1(A); W2(A); R2(B); W2(B); R1(B); W1(B)

Serializable: R1(A); W1(A); R2(B); W2(B); R1(B)

3.-

For security. If the database mustn't modified READ ONLY is the correct access mode. If the database can be modified READ/WRITE is the correct one.

Exercise 18.1 Briefly answer the following questions:

1. How does the recovery manager ensure atomicity of transactions? How does it ensures durability?

2. What is the difference between stable storage and disk?

3. What is the difference between a system crash and a media failure?

4. Explain the WAL protocol.

5. Describe the steal and no-force policies.

1. The Recovery Manager ensures atomicity of transactions by undoing the actions of transactions that do not commit. It ensures durability by making sure that all actions of committed transactions survive system crashes and media failures.
2. Stable storage is guaranteed (with very high probability) to survive crashes and media failures A disk might get corrupted or fail but the stable storage is still expected to retain whatever is stored in it. One of the ways of achieving stable storage is to store the information in a set of disks rather than in a single disk with some information duplicated so that the information is available even if one or two of the disks fail.
3. A system crash happens when the system stops functioning in a normal way or stops altogether. The Recovery Manager and other parts of the DBMS stop functioning (e.g. a core dump caused by a bus error) as opposed to media failure. In a media failure, the system is up and running but a particular entity of the system is not functioning. In this case, the Recovery Manager is still functioning and can start recovering from the failure while the system is still running (e.g., a disk is corrupted.
4. WAL Protocol: Whenever a change is made to a database object, the change is first recorded in the log and the log is written to stable storage before the change is written to disk.
5. a steal policy is in effect, the changes made to an object in the buffer pool by a transaction can be written to disk before the transaction commits. This might be because some other transaction might ”steal” the buffer page presently occupied by an uncommitted transaction.

   A no-force policy is in effect if, when a transaction commits, we need not ensure that all the changes it has made to objects in the buffer pool are immediately forced to disk.

Exercise 18.3

1. The Analysis phase starts with the most recent begin checkpoint record and proceeds forward in the log until the last log record. It determines

(a) The point in the log at which to start the Redo pass

(b) The dirty pages in the buffer pool at the time of the crash.

(c) Transactions that were active at the time of the crash which need to be undone.

The Redo phase follows Analysis and redoes all changes to any page that might have been dirty at the time of the crash. The Undo phase follows Redo and undoes the changes of all transactions that were active at the time of the crash.

2. (a) For this example, we will assume that the Dirty Page Table and Transaction Table were empty before the start of the log. Analysis determines that the last begin checkpoint was at LSN 00 and starts at the corresponding end checkpoint (LSN 10).

We will denote Transaction Table records as (transID, lastLSN) and Dirty Page Table records as (pageID, recLSN) sets.

Then Analysis phase runs until LSN 70, and does the following:

LSN 20 Adds (T1, 20) to TT and (P5, 20) to DPT

LSN 30 Adds (T2, 30) to TT and (P3, 30) to DPT

LSN 40 Changes status of T2 to ”C” from ”U”

LSN 50 Deletes entry for T2 from Transaction Table

LSN 60 Adds (T3, 60) to TT. Does not change P3 entry in DPT

LSN 70 Changes (T1, 20) to (T1, 70)

The final Transaction Table has two entries: (T1, 70), and (T3, 60). The final Dirty Page Table has two entries: (P5, 20), and (P3, 30).

(b) Redo Phase: Redo starts at LSN 20 (smallest recLSN in DPT).

LSN 20 Changes to P5 are redone.

LSN 30 P3 is retrieved and its pageLSN is checked. If the page had been written to disk before the crash (i.e. if pageLSN >= 30), nothing is re-done otherwise the changes are re-done.

LSN 40,50 No action

Changes to P3 are redone

LSN 60

LSN 70 No action

(c) Undo Phase: Undo starts at LSN 70 (highest lastLSN in TT). The Loser Set consists of LSNs 70 and 60. LSN 70: Adds LSN 20 to the Loser Set. Loser Set = (60, 20). LSN 60: Undoes the change on P3 and adds a CLR indicating this Undo. Loser Set = (20). LSN 20: Undoes the change on P5 and adds a CLR indicating this Undo.

Exercise 19.2

1)
The keys are:
{ACD}, {BCD} and {DEC}

2)
Being R the relation schema, for every FD X->A in F, A is part of some key for R:
- B is part of {BCD}
- E is part of {DEC}
- A is part of {ACD}

So R is in 3NF.

3)
Being R the relation schema, for every FD X-> in F, we can find some X's which are not superkey whose A's are not trivial:

Ex.: in BC->E, BC is not a superkey.

So R is not in BCNF

Exercise 19.5

1. 1NF. BCNF decomposition: AB, CD, ACE.

2. 1NF. BCNF decomposition: AB, BF

3. BCNF.

4. BCNF.

5. BCNF.

Exercise 19.6

A	B	C
1	2	3
4	2	3
5	3	3

1.

- *A->B*

The instance satisfies the FD, but we cannot conclude that a FD does hold by looking at one of more instances of the relation.

- *BC->A*

{2,3}->1 and {2,3}->4, the instance violates the FD, so it does not hold over schema S

- *B->C*

The instance satisfies the FD, but we cannot conclude that a FD does hold by looking at one of more instances of the relation.

2.

We can never deduce that an FD does hold by looking at one of more instances of the relation, because an FD, like other ICs, is a statement about all possible legal instances of the relation.

Exercise 19.10

1.

Key: {CD}

We had a 3BNF, and the decomposition generates a collection of BCNF schemas, it is a good decomposition.

2.

Keys: {AB} or {BC}

The decomposition is not preserving the dependency AB->C, so we consider it is not a good decomposition.

3.
Key: {A}

The decomposition is not preserving dependency C->AD, so we consider it is not a good decomposition.

4.
Key: {A}

The decomposition is a lossless decomposition because the common attribute A is key in AB, so we do not think it is a good decomposition.

5.
Key: {A}

The decomposition is not preserving the dependency B->C, so we do not think it is a good decomposition.