Transaction Management

School of Information Sciences, Manipal

(Moonsoon 2016)

Database Transaction

A database transaction encapsulates a sequence of commands that produces effects satisfying the following requirements

- Atomicity
- Consistency
- Isolation
- Durability

ACID Requirements

A DBMS supporting concurrent transactions must ensure

- Atomicity
  A transaction is an all or none unit of effects.
- Consistency
  The state of the database is always consistent.
- Isolation
  There is no interference across transaction boundaries.
- Durability
  The effects are permanent when a transaction commits.

Unmanaged Concurrency - Issues

- Lost update problem
- Uncommitted update problem
- Inconsistent Analysis
- Nonrepeatable Read
- Phantom data

The Lost Update Problem

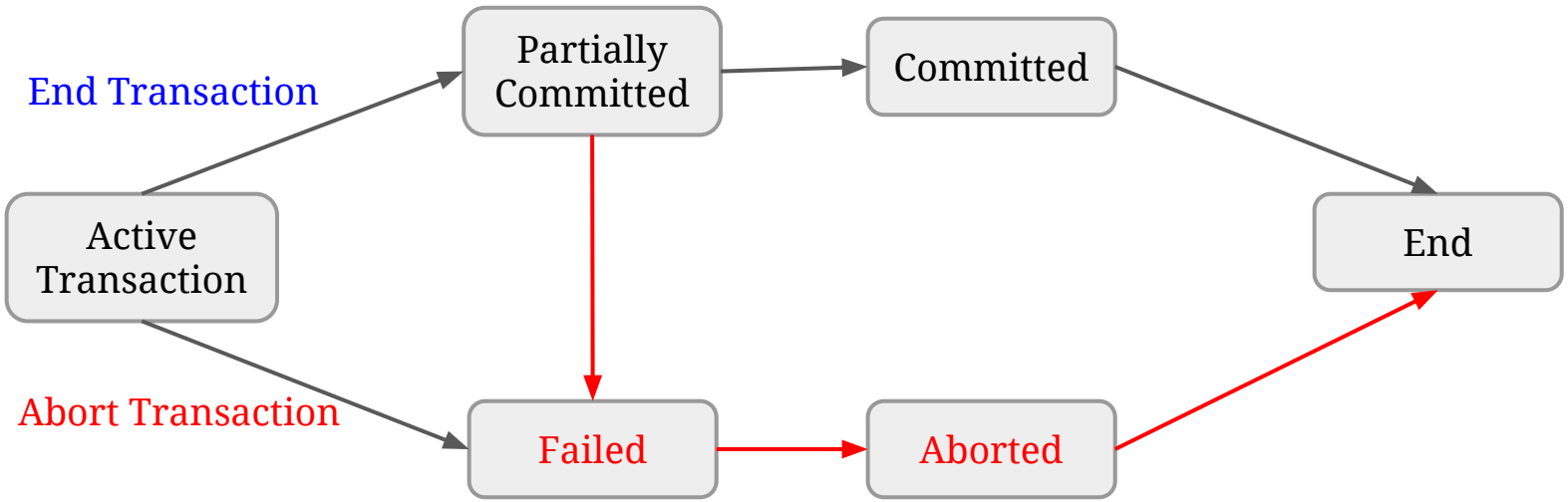
Time	Tx A	Tx B	Balance
t1	BEGIN TX		
t2	read Balance Balance = 1000	BEGIN TX	1000
t3		read Balance Balance = 1000	1000
t4	Balance = Balance - 50 Balance = 950		1000
t5	write Balance Balance = 950	Balance = Balance + 100 Balance = 1100	950
t6	COMMIT		950
t7		write Balance Balance = 1100	1100
t8		COMMIT	1100

The Uncommitted Update Problem

Time	Tx A	Tx B	DB Balance
t1	BEGIN TX		1000
t2	read Balance Balance = 1000		1000
t3	Balance = Balance + 1000 Balance = 2000		1000
t4	write Balance Balance = 2000	BEGIN TX	2000
t5		read Balance Balance = 2000	2000
t6		Balance = Balance + 500 Balance = 2500	2000
t7	ROLLBACK		1000
t8		write Balance Balance = 2500	2500
t9		COMMIT	2500

Time	SUMBAL	Transfer	DB Bal_A	DB Bal_B	DB Bal_C	Sum
t1	BEGIN TX		5000	5000	5000	
t2	Sum = 0	BEGIN TX	5000	5000	5000	
t3	read Bal_A Bal_A = 5000		5000	5000	5000	
t4	Sum = Sum + Bal_A Sum = 5000	read Bal_A Bal_A = 5000	5000	5000	5000	
t5	read Bal_B Bal_B = 5000	Bal_A = Bal_A - 1000 Bal_A = 4000	5000	5000	5000	
t6	Sum = Sum + Bal_B Sum = 10000	write Bal_A Bal_A = 4000	4000	5000	5000	
t7		read Bal_C Bal_C = 5000	4000	5000	5000	
t8		Bal_C = Bal_C + 1000 Bal_C = 6000	4000	5000	5000	
t9		write Bal_C Bal_C = 6000	4000	5000	6000	
t10	read Bal_C Bal_C = 6000	COMMIT	4000	5000	6000	
t11	Sum = Sum + Bal_C Sum = 16000		4000	5000	6000	
t12	write Sum Sum = 16000		4000	5000	6000	16000
t13	COMMIT		4000	5000	6000	16000

Database Transaction State Diagram



Database Recovery and Concurrency Control

Database recovery

process of restoring the database to a correct and consistent state in the event of a failure.

Concurrency control

ability to manage multiple transactions while guaranteeing a correct semantics of non-interference.

Serializability

Serial execution of transactions

- transactions are carried out one after another.
- no interleaving of operations across transactions.

Serializable schedule

- interleaved performance of transactional commands.
- effect is same as *some* serial execution.

Transactions Without Conflicts

Two read-only transactions do not conflict.

Two transactions with no common shared data do not conflict.

If transaction T reads or writes data updated by another transaction S, the order of execution matters.

Conflicting Transactions

Two operations conflict if all of the following is true

- they belong to different transactions.
- they share some data elements.
- at least one of the operations mutates shared data.

Conflict Serializability

Strictest form of serializability

- allow a safe interleaving of conflicting operations.
- result will be same as one the possible serial executions.

DBMS support combination of techniques

- Locking
- Timestamping
- optimistic techniques

Locking and Deadlocks

Most DBMS take an easy route

- no deadlock prevention.

Deadlock detection and recovery

- run deadlock detection periodically.
- choose a victim upon detecting a deadlock.
- avoid *starvation*.

Two Phase Locking Protocol

Acquire phase

- transaction is allowed to obtain required locks.
- *does not* mean acquiring all locks in one go!

Release phase

- no more locks may be acquired.
- only release locks that the transaction owns.

Two Phase Locking Protocol

A transaction must acquire a lock on an item before operating on that item.

- for read-only access, a shared lock is adequate.
- for write access, an exclusive lock is mandatory.

Once the transaction releases a lock, it is not permitted to acquire new locks.

An Example Run of Two Phase Locking Method

Time	Tx A	Tx B	DB Balance
t1	BEGIN TX		
t2	request XLock on Balance		
t3	grant XLock on Balance		
t4	read Balance Balance = 1000	BEGIN TX	1000
t5	Balance = Balance - 50 Balance = 950	request XLock on Balance	1000
t6	write Balance Balance = 950	wait	950
t7	COMMIT Unlock Balance	wait	950
t8		grant XLock on Balance	950
t9		read Balance Balance = 950	950
t10		Balance = Balance + 100 Balance = 1050	950
t11		write Balance Balance = 1050	1050
t12		COMMIT Unlock Balance	1050

Two Phase Locking Protocol

All locks may start as shared locks

- Upgraded to exclusive locks when mutation is required.
- Downgrading happens when locks are released.

Deadlocks are possible in two phase locking methods

- they are usually detected and recovered.
- some transaction(s) may required restart.

Granularity of Locks

Data Item
- element of the set

Tuple
- the “row”

Page
- implementation detail

Table
- the relation

Database
- all relations

Timestamping

No locks are used, and therefore, no deadlocks can occur

Timestamp is an unique id

- defines a temporal partial order over transactions.
- represents the “age” or “generation” of the transaction.

Timestamping

If T is a transaction, TS(T) represents its unique timestamp.

Each data item P gets two timestamps

- read timestamp - RTS(P)
- write timestamp - WTS(P)

RTS(P) the timestamp of the *last transaction* that *read* this data.

RTS(P) the timestamp of the *last transaction* that *updated* this data element.

Basic Timestamping Protocol

Two problems crop up in a timestamped database system.

Late read

- transaction T requests to read an element updated by a younger (newer or later) transaction.

Late write

- Transaction T requests to write an element that has been read or written by a newer transaction.

Basic Timestamping Protocol

If T asks to read a data item P, compare TS(T) with WTS(P)

- if WTS(P) <= TS(T) then
  allow reading
  if RTS(P) < TS(T) then RTS(P) := TS(T) else *skip*

- If WTS(P) > TS(T) then rollback T
  T is attempting a late read.
  The value of P it actually needs has already been updated.

Basic Timestamping Protocol

If T asks to write a data item P, compare TS(T) with both WTS(P) and RTS(P)

- if WTS(P) <= TS(T) and RTS(P) <= TS(T) then
  allow writing
  WTS(P) := TS(T)
  else rollback T

★ If WTS(P) > TS(T), T is trying to write to an obsolete value.
★ If RTS(P) > TS(T), T is trying to update a value already read by a younger transaction. It would be wrong to let this happen.

Multiversion Timestamping

The degree of concurrency may be increased if we allow multiple versions of a data element to be accessible.

In this scheme, a data element P may have a sequence of versions $<P_1, P_2, \ldots, P_n>$

Each version, P_i has a

- content or value field
- WTS(P_i) - timestamp of the last transaction that wrote the value
- RTS(P_i) - timestamp of the youngest transaction that has read version P_i

Multiversion Timestamping Protocol

If T asks to read a data item P

- select version P_i such that WTS(P_i) <= TS(T)

  if RTS(P_i) < TS(T) then RTS(P_i) := TS(T)

Multiversion Timestamping Protocol

If T asks to write a data item P, the version we must use is the one whose WTS is the largest one that is less than or equal to TS(T).

select P_i such that for all $j \neq i$ WTS(P_i) >= WTS(P_j) and WTS(P_i) <= TS(T)
if RTS(P_i) <= TS(T) then
    create a new version P_i' of P_i
    RTS(P_i') := TS(T)
    WTS(P_i') := TS(T)
else
    rollback T

Optimistic Techniques

In many environments, conflicts among transactions is rare.

Conflicts are not common case.

Occasional rollback is tolerable.

It makes sense to optimize for common cases and cope with conflicts on a case-by-case basis.

Optimistic Techniques

Read phase

reads are unchecked; they always succeed.

all writes are performed on local copies of data elements.

Validation phase

if there are no conflicting transactions in the current scope, validation simply succeeds.

Write phase

local copies of data elements are synced with the database.

Optimistic Techniques

Validation phase

Each transaction T is assigned three timestamps.

- start(T) - the starting time of the transaction.
- validation(T) - the time when it enters the validation phase.
- finish(T) - the time it finishes including its write phase.

Optimistic Techniques - Validation Phase

for all transactions S
    validation(S) < validation(T) => finish(S) < start(T)

OR

if start(T) < finish(S) then both of the following should hold:
  data elements written by S are not the ones read by T, and
  finish(S) < validation(T)

Need for Database Transaction Recovery

Disk malfunction.

Failures of various subsystems of the host computer.

Defects in systems software that cause data loss in a DBMS.

Malicious hackers and unintended human errors.

Natural disasters

Recovery Techniques - Basics

Transactions are the primary unit of database recovery.

Recovery manager is the component of a DBMS that is responsible for ensuring atomicity and durability for transactions in the event of failures.

Recovery Techniques - Requirements

The effects of committed transactions must reach the database.

The effects of committed transactions must be permanent.

The effects of any uncommitted transactions must be undone.

Recovery Techniques - Requirements

The effects of committed transactions must reach the database.

The effects of committed transactions must be permanent.

The effects of any uncommitted transactions must be undone.

Stable Storage

- Backups
- Mirrors
  - RAID
  - Remote sites

Recovery - Levels of Data Caches

Application variables.

Database buffer pages.

- not immediately written to storage even for committed transactions.

Database pages.

In a few protocols, updates may reach database even before a commit.

- in case of failures, these changes must be undone.

Recovery Techniques - Transaction Logs

A log file records the state of the system before and after every operation so recovery is possible.

Logically, each record is of the form

- <BEGIN T> start of a transaction T
- <T, x, o, n> attribute name, old and new values.
- <COMMIT T>
- <ABORT T>
- <CHECKPOINT list-of-active-txs> checkpoint status

Recovery - Deferred Update Protocol

DBMS records all writes in the log.

Prepares to write to update the database only when the transaction is ready to commit.

Recovery - Deferred Update Protocol

When the transaction starts

write <BEGIN T> to the log

When data is mutated

write <T, x, o, n> to the log

When transaction is ready to commit

write <COMMIT T>

write the log to the disk

update the database using records in the log

Recovery - Deferred Update Protocol

When the transaction starts
    write <BEGIN T> to the log

When data is mutated
    write <T, x, o, n> to the log

When transaction is ready to commit
    write <COMMIT T>
    write the log to the disk
    update the database using records in the log

If the transaction aborts, ignore the log reecords
    do not perform any writes to the database.

Recovery - Deferred Update Protocol

Checkpoints run at regular intervals

force write modified pages in the database buffers to disk.

write log records in memory out to disk.

write a checkpoint record to the log.

- consists of a list of active transactions at this moment.