Transactions

Transaction: an execution of a user program in a DBMS.

Transaction properties:

- **A**tomicity: all-or-nothing execution
- **C**onsistency: database consistency is preserved
- **I**solation: concurrently executing transactions have no effect on one another
- **D**urability: results survive failures.

Schedules

Transaction (DBMS view):

- list of actions (**read** or **write**)
- terminated by **commit** or **abort**

Schedule:

- interleaving of multiple transactions
- action order within transaction preserved
- **complete:** commit/abort for every transaction
- **serial:** no interleaving of actions from different transactions
- **serializable:** equivalent to a serial schedule (assuming all transactions commit).

Conflicts

Conflict:

- a pair of actions on the same data object
- one action is a write.

Types of conflict:

- **WR:** reading uncommitted data
- **RW:** unrepeatable reads
- **WW:** overwriting uncommitted data.

Reading uncommitted data

T_1	debit(A,1000), credit(B,1000)
T_2	increase A by 10%, increase B by 10%

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
R(B)	
W(B)	
Commit	

Unrepeatable read

T_3	credit(A,1000)
T_4	credit(A,2000)

T_3	T_4
R(A)	
	R(A)
	W(A)
	Commit
W(A)	
Commit	

Overwriting uncommitted data

T_5	book(F1,AA), book(F2,AA)
T_6	book(F1,Delta), book(F2,Delta)

T_5	T_6
W(F1)	
	W(F1)
	W(F2)
	Commit
W(F2)	
Commit	

Aborted transactions

The effect of aborted transactions has to be completely undone.

Problems:

- a transaction depending on an aborted transaction may have already committed (**unrecoverable** schedule)
- aborting a transaction requires aborting other transactions (**cascading aborts**).
- undoing the actions of an aborted transaction by restoring original values may erase subsequent updates of committed transactions.

Strict two-phase locking

Rules:

1. A **read object** action is immediately preceded by a **shared** lock request on the object, a **write object** action by an **exclusive** lock request on the object.
2. all the locks are held until the end of transaction.

Guarantees:

- schedule serializability
- schedule recoverability
- no cascading aborts
- restoring old values does not erase subsequent updates.

Locking

Multiple transactions can hold **shared** locks on the same object, but only one transaction can hold an **exclusive** lock on an object.

Locks are stored in a **lock table** (managed by DBMS), lock requests are **queued**.

Lock/unlock: atomic operations.

Problems:

- deadlocks
- starvation.

Deadlocks

A set of transactions such that each **waits for** a lock held by another one.

Handling deadlocks:

- **prevention:**
  - transaction priorities
  - obtaining all the locks at the beginning
- **detection:**
  - identifying cycles in the waits-for graph or timeout, and
  - abort transaction.

Handling starvation:

- FIFO lock queues.

Database recovery

Types of failures:

- transaction failures
- system crashes
- media failures

Log of all database modifications in **stable storage**.

Basics

Atomic writes: writing a page to disk is an atomic action.

Steal: writing some transaction changes to disk before it commits.

Force: writing all transaction changes right after it commits.

No-steal/force, or steal/no-force?

Aries

Steal/no-force approach, used in IBM DB2.

Phases:

1. Analysis
2. Redo
3. Undo

Basic principles:

1. Write-Ahead Logging
2. Repeating History during Redo
3. Logging Changes during Undo

Distributed transactions

Transactions:

- **subtransactions** executing at different sites
- all subtransactions commit or none does (**commit protocol**)
- site and link failures.

Two-phase commit

A site is designated as a **coordinator**, other participating sites are **subordinates**.

Protocol:

1. *Coordinator:* send a PREPARE message to each subordinate
2. *Subordinate:* receive PREPARE and decide to commit or abort:
   - **commit:** force-write a prepare log record, reply YES
   - **abort:** force-write an abort log record, reply NO

3. *Coordinator:*
   - all subordinates reply YES: force-write a commit log record, send a COMMIT message to each subordinate
   - one replies NO or times out: force-write an abort log record, send an ABORT message to each subordinate
4. *Subordinate:*
   - receive COMMIT: force-write a commit log record, send ACK to coordinator, commit
   - receive ABORT: force-write an **abort** log record, send ACK, abort
5. *Coordinator:* receive ACK from all subordinates, write end log record.