Transactions

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

Transaction properties:

- Atomicity: all-or-nothing execution
- Consistency: database consistency is preserved
- Isolation: concurrently executing transactions have no effect on one another
- Durability: 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**: unrepeateable reads
- **WW**: overwriting uncommitted data.
## Reading uncommitted data

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>debit($A,1000$), credit($B,1000$)</td>
</tr>
<tr>
<td>$T_2$</td>
<td>increase $A$ by 10%, increase $B$ by 10%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>R($A$)</td>
<td>R($A$)</td>
</tr>
<tr>
<td>W($A$)</td>
<td>W($A$)</td>
</tr>
<tr>
<td></td>
<td>R($B$)</td>
</tr>
<tr>
<td></td>
<td>W($B$)</td>
</tr>
<tr>
<td></td>
<td>Commit</td>
</tr>
<tr>
<td>R($B$)</td>
<td></td>
</tr>
<tr>
<td>W($B$)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Commit</td>
</tr>
</tbody>
</table>
## Unrepeatable read

<table>
<thead>
<tr>
<th>$T_3$</th>
<th>credit($A, 1000$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_4$</td>
<td>credit($A, 2000$)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$T_3$</th>
<th>$T_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R(A)$</td>
<td>$R(A)$</td>
</tr>
<tr>
<td>$W(A)$</td>
<td>$W(A)$</td>
</tr>
<tr>
<td>Commit</td>
<td>Commit</td>
</tr>
</tbody>
</table>
Overwriting uncommitted data

<table>
<thead>
<tr>
<th>$T_5$</th>
<th>book(F1,AA), book(F2,AA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_6$</td>
<td>book(F1,Delta), book(F2,Delta)</td>
</tr>
</tbody>
</table>

\[
\begin{array}{c|c}
T_5 & T_6 \\
------------------- & \\
W(F1) & W(F1) \\
 & W(F1) \\
 & W(F2) \\
 & Commit \\
W(F2) & Commit \\
\end{array}
\]
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 \textit{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.