**ARINDA HILLARY**

**DBMS**

**ASSIGNMENT 7: 2PL and Deadlock**

**3/12/2024**

**(1)** [3]**For all the examples you created in earlier Assignment 6 - Q2, show in detail how 2PL solves all those problems.**

1. **Lost Update Problem:**

**Problem Example**: Consider a bank account with a balance of $500. User A starts a transaction to debit $100, and at the same time, User B initiates a transaction to credit $200. Both read the initial balance as $500. User A calculates the new balance as $400 and writes it back. Shortly after, User B calculates the new balance as $700 and writes it back. The debit of $100 from User A's transaction is lost.

| **Time** | **Transaction A (TA)** | **Transaction B (TB)** | **Balance x** |
| --- | --- | --- | --- |
| t1 | Begin transaction |  |  |
| t2 | read(balx) = $500 | Begin transaction | $500 |
| t3 | balx = balx - $100 | read(balx) = $500 | $500 |
| t4 | write(balx) = $400 | balx = balx + $200 | $400 |
| t5 | commit | write(balx) = $700 | $700 |
| t6 |  | commit | $700 |

**Solution:**

| **Time** | **Transaction A (TA)** | **Transaction B (TB)** | **Balance x** |
| --- | --- | --- | --- |
| t1 | Begin transaction |  |  |
| t2 | Write\_lock(balx) | Begin transaction | $500 |
| t3 | read(balx) = $500 | Write\_lock(balx) | $500 |
| t4 | balx = balx - $100 | WAIT | $500 |
| t5 | write(balx) = $400 | WAIT | $400 |
| t6 | Commit/unlock(balx) | WAIT | $400 |
| t7 |  | read(balx) = $400 | $400 |
| t8 |  | balx = balx + $200 | $400 |
| t9 |  | write(balx) = $600 | $600 |
| t10 |  | Commit/unlock(balx) | $600 |

**Explanation:**

t1: User A starts a transaction, immediately requesting and obtaining a write lock on the account balance, preventing other transactions from concurrently modifying it.

t2: Holding the lock, User A reads the balance from the account, which is $500 at this point.

t3: User A proceeds to deduct $100 from the balance, updating the balance to $400.

t4: After the update, User A commits the transaction, which finalizes the changes. Subsequently, the write lock is released, making the new balance available for other transactions.

t5: User B now initiates their transaction. However, due to 2PL, they had to wait until User A's lock was released before proceeding.

t6: Once the lock is available, User B acquires it and reads the committed balance, which reflects User A's deduction and is now at $400.

t7: User B adds $200 to the current balance, which leads to an updated balance of $600.

t8: User B commits their transaction, which solidifies the addition to the account balance.

t9: Finally, with the commit complete, User B releases the write lock on the account balance.

t10: The account balance now stands at $600, accurately reflecting the transactions from both User A and User B due to the serialized access enforced by 2PL.

1. **Uncommitted Dependency / Dirty Read:**

**Problem Example**: User A initiates a transaction to deposit $300 into their checking account but hasn't committed the transaction yet. The balance temporarily reflects the $300 addition. Meanwhile, User B, in a different session, views the balance of User A’s checking account and sees the uncommitted addition of $300. Acting on this information, User B initiates a wire transfer of $300 from User A's account to an external account. However, if User A's transaction is later rolled back due to an error or another issue, the $300 that User B saw and transferred was never officially recorded, leading to an overdraft when the wire transfer is processed.

| **Time** | **Transaction A (TA)** | **Transaction B (TB)** | **Balance x (checking)** |
| --- | --- | --- | --- |
| t1 | begin\_transaction |  | $500 |
| t2 | Read(balx) |  | $500 |
| t3 | bal = bal + $300 (uncommitted) |  | $500 |
| t4 | Write(balx) | Begin transaction | $800 (uncommitted) |
| t5 |  | views uncommitted balance = $800 (readx) | $800 (uncommitted) |
| t6 | rollback complete, deposit not made | initiates wire transfer of $300 balx=balx-300 | $800 |
| t7 |  | wire transfer processed (write (balx) | $500 |
| t8 |  | commit | $500 |

In this table, the uncommitted balance of $800 is not the actual balance due to the rollback of Transaction A. Transaction B, however, has initiated a transfer based on this incorrect balance, leading to a post-transfer balance that would actually be an overdraft**.**

**Solution**

| **Time** | **Transaction A (TA)** | **Transaction B (TB)** | **Balance x (checking)** |
| --- | --- | --- | --- |
| t1 | begin\_transaction |  | $500 |
| t2 | Write\_lock(balx) |  | $500 |
| t3 | Read(balx) |  | $500 |
| t4 | bal = bal + $300 | Begin transaction | $500 |
| t5 | Write(balx) | Write\_lock(balx) | $800 |
| t6 | rollback complete, deposit not made/unlock(balx) | WAIT | $500 |
| t7 |  | initiates wire transfer of $300 balx=balx-300 | $500 |
| t8 |  | wire transfer processed (write (balx) | $200 |
| t9 |  | Commit/unlock(balx) | $200 |

**Explanation**

t1: Transaction A begins and acquires a write lock on the account balance, preventing other transactions from accessing the balance while Transaction A's operations are underway.

t2: Holding the lock, Transaction A reads the account balance, which is $500.

t3: Transaction A increases the balance by $300, reflecting an uncommitted balance of $800.

t4: Before Transaction A can commit, Transaction B begins and attempts to acquire a write lock on the same account balance. However, Transaction B must wait because Transaction A's lock is still in place.

t5: Transaction A detects an issue and rolls back the changes, reverting the balance to the original $500. Transaction A then releases the lock.

t6: Transaction B now acquires the lock and initiates a wire transfer of $300, reducing the balance to $200.

t7: The wire transfer is processed, and Transaction B writes the new balance ($200) to the account.

t8: Transaction B commits the transaction and releases the lock.

t9: The final committed balance in the account is $200, and no dirty read has occurred because Transaction B only read and acted on committed data.

1. **Inconsistent Analysis Problem:**

**Problem Example**: User A is generating a report based on total sales for the month. The report runs in multiple steps and reads the total sales amount more than once. Meanwhile, User B enters a new sale and commits it to the database after User A's first read but before the second read. This results in User A's report reflecting an inconsistent total sales amount because the data has changed during the report generation process.

| **Time** | **Transaction A (TA)** | **Transaction B (TB)** | **Total Sales** |
| --- | --- | --- | --- |
| t1 | Begin transaction |  |  |
| t2 | start\_report: read(total\_sales) | Begin transaction | $2000 |
| t3 | continue\_report (no read yet) | enter\_sale + $500 | $2000 |
| t4 |  | write(total\_sales) = $2500 | $2500 |
| t5 | read(total\_sales) = $2500 (updated) | commit | $2500 |
| t6 | commit |  | $2500 |

**Solution**

| **Time** | **Transaction A (TA)** | **Transaction B (TB)** | **Total Sales** |
| --- | --- | --- | --- |
| t1 | begin\_transaction |  | $2000 |
| t2 | Read\_lock(total\_sales) |  | $2000 |
| t3 | start\_report: read(total\_sales) | Begin transaction | $2000 |
| t4 |  | Write\_lock(total\_sales) | $2000 |
| t5 | continue\_report | WAIT | $2000 |
| t6 |  | enter\_sale + $500 | $2000 |
| t7 | read(total\_sales) = $2000 | WAIT | $2000 |
| t8 | Commit/unlock(total\_sales) |  | $2000 |
| t9 |  | write(total\_sales) = $2500 | $2500 |
| t10 |  | Commit/unlock(total\_sales) | $2500 |

t1: Transaction A begins and acquires a read lock on total\_sales.

t2: Transaction A reads total\_sales under the read lock, ensuring that the value of total\_sales cannot change while it holds the lock.

t3: Transaction B begins and tries to acquire a write lock on total\_sales, but it must wait because Transaction A holds a read lock on the same data.

t4: Transaction A continues to work on the report, still within the protected read lock session.

t5: Transaction B remains in the waiting state until Transaction A releases the read lock.

t6: Transaction A reads total\_sales again, if necessary, and the value is still $2000 due to the read lock.

t7: Transaction A commits, ending its transaction, and releases the read lock on total\_sales.

t8: Transaction B acquires the write lock on total\_sales after the read lock is released.

t9: Transaction B writes the new total\_sales value ($2500) to the database.

t10: Transaction B commits its transaction and releases the write lock on total\_sales.

**(2)**[4]**Consider following sequence of actions, listed in the order the actions are presented to the DBMS.**

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

**Assume that the concurrency control mechanism is 2PL with “Wound-Wait” deadlock prevention strategy.**  
**Acquire locks as late as possible and release locks as early as possible. Waiting transactions continued and brought up to date as early as possible.**

**Describe how the 2PL concurrency control mechanism handles the sequence of actions.**

**t1- T1 wants shared lock on X and gets it -> T1:R(X)**

t2-T2 wants and exclusive lock on X but since T2 is younger than T1, T2 waits for T1

t3-T3 wants exclusive lock on Y and gest it -> **T3:W(Y)**

t4-T1 wants an exclusive lock on Y and since T1 is older than T3, T3 is aborted and T1 gets the exclusive lock on Y

t5-T1 releases the lock on X

t6-T2 gets exclusive lock on X->**T2:W(X)**

t7-**T1:W(Y)**

t8- T1 commits and releases the lock on Y

t9-T2 wants exclusive lock on Y and gets it -> **T2:W(Y)**

t10-T2 commits and releases the lock on Y

t11-T3 restarts and gets exclusive lock on Y-**T3:W(Y)**

t12-T3 wants an exclusive lock on Z and gets it –> **T3:R(Z)**

t13-**T3:W(Z)**

t14-T3 commits and releases lock on Z

**(3)**[3] **Produce a wait-for-graph for the following transaction scenario and determine whether deadlock exists.**

|  |  |  |
| --- | --- | --- |
| **Transaction** | **Data items locked by Transaction** | **Data items transaction is waiting for** |
| **T1** | x2 | x1, x3 |
| **T2** | x3, x10 | x7, x8 |
| **T3** | x8 | x4, x5 |
| **T4** | x7 | x1 |
| **T5** | x1, x5 | x3 |
| **T6** | x4, x9 | x6 |
| **T7** | x6 | x5 |

A diagram of a network

Description automatically generated with medium confidence

**Existing deadlocks**

* **T2,T4,T5**
* **T2,T3,T5**
* **T2,T3,T6,T7,T5**