**ARINDA HILLARY**

**DBMS**

**ASSIGNMENT 8: Timestamping**

**3/14/2024**

1. [2] **What is meant by granularity? Give examples.**

Granularity is the size of data items chosen as a protection unit by concurrency control protocol.  
 **Examples**

1. Entire Database Granularity:

Example: In a banking system, a database administrator may lock the entire database during a major update or system migration to prevent any transactions from occurring, ensuring the consistency and integrity of the entire database.

1. Table Granularity:

Example: When applying an update to all customer records in a customers table, a table lock might be used. This would prevent other transactions from reading or writing to the customers table until the lock is released.

1. Page-Level Granularity:

Example: A page lock could be applied when a transaction is working with a specific set of rows that are stored in the same page on disk. If a transaction needs to update several records that reside on the same page, it's more efficient to lock the entire page than to lock each row individually.

1. Row-Level Granularity:

Example: If an application is updating a user's profile information, a row-level lock on just that user's row in the user profiles table would suffice. This allows other transactions to concurrently update other users' profiles.

1. Column-Level Granularity:

Example: A transaction that calculates the new balance for a bank account after a deposit may only need to lock the balance column of a row in the accounts table. While this level of granularity is rare due to the overhead of such fine-grained locks, it's theoretically possible in certain systems.

1. [3]**Give full details of a timestamping mechanism for concurrency control that can be used to ensure that the types of problems discussed in Assignment6-Q2 cannot occur.**   
   Show how the timestamping mechanism prevents the problems illustrated from occurring.  
   You can use any of the two timestamping methods we studied in the lecture.

**Using single version timestamping**

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** | **Action** |
| --- | --- | --- | --- | --- |
| t1 | ts(TA) = 100 |  | $500 | TA begins |
| t2 | read(balx) | ts(TB) = 200 | $500 | TB begins |
| t3 | balx = balx - $100 |  | $500 | TA calculates new balx |
| t4 | write(balx) = $400 |  | $400 | TA writes new balx |
| t5 | commit |  | $400 | TA commits |
| t6 |  | read(balx) | $400 | TB reads committed balx by TA |
| t7 |  | balx = balx + $200 | $400 | TB calculates new balx |
| t8 |  | write(balx) = $600 | $600 | TB writes new balx |
| t9 |  | commit | $600 | TB commits |

**Explanation**

* TA starts with timestamp ts(TA).
* TB starts with timestamp ts(TB) where ts(TB) > ts(TA).
* When TA reads balx, it timestamps this read operation with ts(TA).
* TA writes balx = $400 with timestamp ts(TA).
* TB tries to read balx, but since ts(TB) is greater than the write timestamp of balx by TA, it must wait until TA is committed or aborted.
* TA commits.
* TB can now read the new balance of $400, add $200, resulting in a correct balance of $600.
* TB commits the new balance of $600.

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)** | **Action** |
| --- | --- | --- | --- | --- |
| t1 | ts(TA) = 100 |  | $500 | TA begins |
| t2 | read(balx) |  | $500 | TA reads balx |
| t3 | bal = balx + $300 |  | $800\* | TA writes uncommitted balx |
| t4 |  | ts(TB) = 200 | $500 | TB begins after TA |
| t5 |  | read(balx) | $500 | TB reads committed balx |
| t6 | rollback |  | $500 | TA rollbacks |
| t7 |  | balx = balx - $300 | $500 | TB calculates new balx |
| t8 |  | commit | $200 | TB commits |

**Explanation**

* TA starts and writes to balx with ts(TA) but does not commit.
* TB starts and tries to read balx.
* Since the last write operation on balx by TA has not been committed, TB is not allowed to read this value.
* TB must wait until TA commits or aborts to read the value of balx.
* TA rollbacks, and the write operation is not committed.
* TB reads the correct balance of $500 since TA’s changes were not committed.

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** | **Action** |
| --- | --- | --- | --- | --- |
| t1 | ts(TA) = 100 |  | $2000 | TA begins |
| t2 | start report | ts(TB) = 200 | $2000 | TB begins |
| t3 | read(total\_sales) |  | $2000 | TA reads total\_sales |
| t4 |  | bal = total\_sales + $500 | $2000 | TB calculates new total\_sales |
| t5 |  | write(total\_sales) = $2500 | $2500\* | TB writes new total\_sales |
| t6 | continue report | commit | $2500 | TB commits |
| t7 | read(total\_sales) |  | $2000 | TA reads consistent total\_sales |
| t8 | commit |  | $2000 | TA commits report |

**Explanation**

* TA starts to generate a report and reads total\_sales with ts(TA).
* TB starts and writes a new total\_sales value with ts(TB).
* Since TB has a later timestamp, its write will not affect TA’s read operation, and TA can continue with a consistent view of data.
* TB commits.
* TA reads total\_sales again. Since the read timestamp ts(TA) is less than the write timestamp ts(TB), TA will read the old value of total\_sales for consistency.

1. [5]**Consider the following non-serial schedule.  Show how this schedule will get executed with single version & multi-version timestamping.  Show all the detailed steps.**

**Start(T1), Write(T1, x),  Start(T2),  Read(T2, x),  Start(T3),  Write(T2, x),  Write(T3, x), Read(T2, x),  Write(T2, x),  Commit(T1),  Commit(T2),  Commit(T3)**

**Single version time stamping**

|  |  |
| --- | --- |
| **S Operations** | **Variable (readTS, writeTS)** |
| (t1, start(T1)) |  |
| (t2, write(T1,x)) | X(\_ , t1) |
| (t3,start(T2)) |  |
| (t4,Read(T2,x)) | X(t3 , t1) |
| (t5,start(T3)) |  |
| (t6,Write(T2,x)) | X(t3, t3) |
| (t7, Write(T3, x)) | X(t3, t5) |
| (t8, Read(T2, x)) | **Abort T2** |
| (t9, Commit(T1)) |  |
| (t9,start(T2) |  |
| (t10, Commit (T3)) |  |
| (t11, Read(T2,x) | X(t9,t5) |
| (t12, Write(T2,x)) | X(t9,t9) |
| (t13, Read(T2,x)) | X(t9,t9) |
| (t14, Write(T2,x)) | X(t9,t9) |
| (t15, Commit(T2)) |  |

**Start(T1), Write(T1, x),  Start(T2),  Read(T2, x),  Start(T3),  Write(T2, x),  Write(T3, x), Read(T2, x),  Write(T2, x),  Commit(T1),  Commit(T2),  Commit(T3)**

**Multi-version time stamping**

|  |  |
| --- | --- |
| **Operation of S** | **VARversion (readTS, writeTS)** |
| (t1, Start(T1)) | //Assume X0(t0,t0) |
| (t2, Write (T1, X)) | X1(t1, t1) |
| (t3, Start(T2)) |  |
| (t4, Read (T2, x)) | X1(t3, t1) |
| (t5, Start(T3)) |  |
| (t6, Write (T2, x)) | X2(t3, t3) |
| (t7, Write (T3, x)) | X3(t5, t5) |
| (t8, Read (T2, x)) | X2(t3, t3) |
| (t9, Write (T2, x)) | X4(t3, t3) |
| (t10, Commit(T1)) |  |
| (t11, Commit(T2)) |  |
| (t12, Commit(T3)) |  |