**ARINDA HILLARY**

**DBMS**

**ASSIGNMENT 6: Transactions- Serializability and Recoverability**

**3/11/2024**

1. **[2] Explain what is meant by a transaction. Why are transactions important units of operation in a DBMS?**

A transaction is an action of series of actions carried out by a single user or application program that reads of updates the contents of a database.

Transactions are important units of operation in a DBMS because.

* They ensure data integrity: By treating multiple steps as a single unit, a transaction ensures that either all steps are completed, and the database remains consistent, or if any step fails, none are applied, preventing partial updates that can leave the database in an inconsistent state.
* They provide isolation among concurrent users: The isolation property prevents one user's transaction from interfering with another's, ensuring data correctness when multiple users are accessing the database concurrently.
* They help with recovery: In case of a system failure, the ACID properties of transactions allow the system to roll back to a known good state, which simplifies recovery.
* Transactions thus enable a reliable, predictable management of data within a DBMS.

1. [3]**Describe, with your own examples, the types of problems that can occur in a multi-user environment when concurrent access to the database is allowed.**
2. **Lost Update Problem:**

**Definition**: This problem occurs when two transactions that access the same data are interleaved in a way that one transaction overwrites the data changes of the other transaction, causing the first transaction's changes to be lost.

**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 |

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

**Definition**: A dirty read occurs when a transaction reads data that has been written by another transaction that has not yet been committed. If the second transaction is rolled back, the first transaction has read and may act upon data that was never confirmed.

**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**.**

1. **Inconsistent Analysis Problem:**

**Definition**: Also known as non-repeatable reads, this problem occurs when a transaction reads the same row multiple times and each time retrieves different data, because other transactions are updating the row between the reads.

**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 |

**3) [**2.5]**For each of the following schedules, state whether the schedule is conflict serializable**(using the swapping technique)**and recoverable.**  
For the purposes of finding conflict-serializability, you can assume that instead of abort, it was commit.

1. read(T1, balx), read(T2, balx), write(T1, balx), write(T2, balx), commit(T1), commit(T2)

read(T2, balx), read(T1, balx), write(T1, balx), write(T2, balx), commit(T1), commit(T2)

conflict serializable: **non conflict serializable**

recoverability: **recoverable**

1. read(T1, balx), read(T2, baly), write(T3, balx), read(T2, balx), read(T1, baly), commit(T1), commit(T2)

read(T1, balx), read(T2, baly), write(T3, balx), read(T1, baly), read(T2, balx), commit(T1), commit(T2)

read(T1, balx), read(T2, baly), read(T1, baly), write(T3, balx), read(T2, balx), commit(T1), commit(T2)

read(T1, balx), read(T1, baly), read(T2, baly), write(T3, balx), read(T2, balx), commit(T1), commit(T2)

read(T1, balx), read(T1, baly), write(T3, balx), read(T2, baly), read(T2, balx), commit(T1), commit(T2)

conflict serializable: **conflict serializable ->T1,T3,T2**

recoverability: **With the information provided, we cannot definitively state that it is recoverable, as the commit order for T3 is not given. If T3 commits after T2, the schedule would not be recoverable. If it commits before T2, then it would be recoverable.**

1. read(T1, balx), write(T2, balx), write(T1, balx), abort(T2), commit(T1)

conflict serializable: **non** **conflict serializable**

recoverability: **recoverable**

1. write(T1, balx), read(T2, balx), write(T1, balx), commit(T2), abort(T1)

conflict serializable: **non** **conflict serializable**

recoverability: **non recoverable**

1. read(T1, balx), write(T2, balx), write(T1, balx), read(T3, balx), commit(T1), commit(T2), commit(T3)

conflict serializable: **non** **conflict serializable**

recoverability: **recoverable**

**4)**[2.5]**Draw a precedence graph for each of the schedules (a) – (e) in the previous exercise and state whether the schedule is conflict serializable from the graph.**  
For the purposes of finding conflict-serializability, you can assume that instead of abort, it was commit.

1. read(T1, balx), read(T2, balx), write(T1, balx), write(T2, balx), commit(T1), commit(T2)

A diagram of a circle with arrows

Description automatically generated

conflict serializable: **non conflict serializable**

Precedence graph has a cycle, so the given schedule is non-conflict serializable schedule.

1. read(T1, balx), read(T2, baly), write(T3, balx), read(T2, balx), read(T1, baly), commit(T1), commit(T2)

A diagram of a diagram

Description automatically generated

Precedence graph has no cycle, so the given schedule is a conflict serializable schedule.

1. read(T1, balx), write(T2, balx), write(T1, balx), abort(T2), commit(T1)

A black and white image of a black and white image of a black and white image of a black and white image of a black and white image of a black and white image of a black and

Description automatically generated

Precedence graph has a cycle, so the given schedule is non-conflict serializable schedule.

1. write(T1, balx), read(T2, balx), write(T1, balx), commit(T2), abort(T1)

A black and white image of a black and white image of a black and white image of a black and white image of a black and white image of a black and white image of a black and

Description automatically generated

Precedence graph has a cycle, so the given schedule is non-conflict serializable schedule.

1. read(T1, balx), write(T2, balx), write(T1, balx), read(T3, balx), commit(T1), commit(T2), commit(T3)

A diagram of a triangle with arrows and circles

Description automatically generated

Precedence graph has a cycle, so the given schedule is non-conflict serializable schedule.