Lab 8

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

Ans:-

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

* The primary benefit of using transactions is data integrity. Many database uses require storing data to multiple tables, or multiple rows to the same table in order to maintain a consistent data set. Using transactions ensures that other connections to the same database see either all the updates or none of them.
* A secondary benefit of using transactions is speed. There is often an overhead associated with actually committing the data to the database.

1. **[2] 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.**

**Ans:-**

An apparently successfully completed update operation by one user can be overridden by another user.

Example: T1 Online shopping of item that costs $100 from Walmart

T2 Online shopping of other item that costs $50 from Walmart

The final shopping cost would be $150 no matter which transaction is performed first.

|  |  |  |  |
| --- | --- | --- | --- |
| Time | T1 | T2 | Totalp |
| t1 | begin\_transaction |  | 0 |
| t2 | read(Totalp) | begin \_transaction 0 | |
| t3 | Totalp += 100 | read(Totalp) | 0 |
| t4 | write(Totalp) | Totalp += 50 | 100 |

|  |  |  |
| --- | --- | --- |
| t5 comm | it write(Totalp) | 50 |
| t6 | commit | 50 |

**Uncommitted dependency problem**

Occurs when one transaction is allowed to see the intermediate results of another transaction before it has committed.

T3 updates Totalp to $100 but it aborts,so Totalp should be back at original value of $0.

T4 has read new value Totalp $100 and uses the value to add $50 and give a new Totalp $150, instead of $50

|  |  |  |  |
| --- | --- | --- | --- |
| Time | T3 | T4 | Totalp |
| t1 | begin\_transaction |  | 0 |
| t2 | read(Totalp) |  | 0 |
| t3 | Totalp += 100 |  | 0 |
| t4 | write(Totalp) | begin \_transaction | 100 |
| t5 |  | read(Totalp) | 100 |
| t6 | rollback | Totalp += 50 | 0 |
| t7 |  | write(Totalp) | 150 |
| t8 |  | commit | 150 |

**Inconsistent analysis problem**

Occurs when transaction reads several values but second transaction updates some of them during execution of first.

T5 is total shopping price of Totalp $50 and Totalp $100

T6 has shopped $50 from Totalp shopping cart and returned $50 from Totalq shopping cart, so T5 now has wrong result($50 to low)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time | T5 | T6 | Totalp | Totalq | total |
| t1 | begin\_transaction |  | 50 | 100 |  |
| t2 | total = 0 | begin\_transaction | 50 | 100 | 0 |
| t3 | read(Totalp) | read(Totalp) | 50 | 100 | 0 |
| t4 | total += Totalp | Totalp += 50 | 50 | 100 | 50 |
| t5 |  | write(Totalp) | 100 | 100 | 50 |
| t6 |  | read(Totalq) | 100 | 100 | 50 |
| t7 |  | Totalq -= 50 | 100 | 100 | 50 |
| t8 |  | write(Totalq) | 100 | 50 | 50 |
| t9 | read(Totalq) | commit | 100 | 50 | 50 |
| t10 | total += Totalq |  | 100 | 50 | 100 |
| t11 | commit |  | 100 | 50 | 100 |

1. **[2] For all the examples you created in Q2, show in details how 2PL solves the problem.**  
     Ans:-

**Preventing Lost update problem using 2PL**

Time T1 T2 Totalp t1 begin\_transaction 0

|  |  |  |  |
| --- | --- | --- | --- |
| t2 | write\_lock(Totalp) | begin \_transaction | 0 |
| t3 | read(Totalp) | write\_lock(Totalp) | 0 |
| t4 | Totalp += 100 | **WAIT** | 0 |
| t5 | write(Totalp) | **WAIT** | 100 |
| t6 | commit | **WAIT** | 100 |

|  |  |  |
| --- | --- | --- |
| t7 | read(Totalp) | 100 |
| t8 | Totalp += 50 | 100 |
| t9 | write(Totalp) | 150 |
| t10 | commit | 150 |

**Preventing Uncommitted Dependency Problem using 2PL**

|  |  |  |  |
| --- | --- | --- | --- |
| Time | T3 | T4 | Totalp |
| t1 | begin\_transaction |  | 0 |
| t2 | write\_lock(Totalp) |  | 0 |
| t3 | read(Totalp) |  | 0 |
| t4 | Totalp += 100 | begin \_transaction | 0 |
| t5 | write(Totalp) | write\_lock(Totalp) | 100 |
| t6 | rollback | **WAIT** | 0 |
| t7 |  | read(Totalp) | 0 |
| t8 |  | Totalp += 50 | 0 |
| t9 |  | write(Totalp) | 50 |
| t10 |  | commit | 50 |

**Preventing Inconsistent Analysis Problem using 2PL**

Time T5 T6 Totalp Totalq total

t1 begin\_transaction 50 100

t2 total = 0

|  |  |  |  |
| --- | --- | --- | --- |
| begin\_transaction | 50 | 100 | 0 |
| write\_lock(Totalp) | 50 | 100 | 0 |
| read(Totalp) | 50 | 100 | 0 |

t3

t4 read\_lock(Totalp)

t5 **WAIT** Totalp += 50 50 100 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| t6 | **WAIT** | write(Totalp) | 100 | 100 | 0 |
| t7 | **WAIT** | write\_lock(Totalq) | 100 | 100 | 0 |
| t8 | **WAIT** | read(Totalq) | 100 | 100 | 0 |
| t9 | **WAIT** | Totalq -= 50 | 100 | 100 | 0 |
| t10 | **WAIT** | write(Totalq) | 100 | 50 | 0 |
| t11 | **WAIT** | commit 100 | | 50 | 0 |
| t11 | read(Totalp) | 100 | | 50 | 0 |
| t11 | total += read(Totalp) | 100 | | 50 | 100 |
| t11 | read(Totalq) | 100 | | 50 | 100 |
| t11 | total += Totalq | 100 | | 50 | 150 |
| t11 | commit | 100 | | 50 | 150 |

1. **[2.5] For each of the following schedules, state whether the schedule is conflict serializable, recoverable and whether it avoids cascading aborts.**
2. read(T1, balx), read(T2, balx), write(T1, balx), write(T2, balx), commit(T1), commit(T2)

Ans:- Not -conflict serializable

* + Recoverable
  + Avoid cascading abort Schedule

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

Ans:- Conflict serializable

* + non-Recoverable
  + Not Avoid cascading abort Schedule

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

Ans:- Conflict serializable

* + Recoverable
  + Avoid cascading abort Schedule

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

Ans:- Conflict serializable

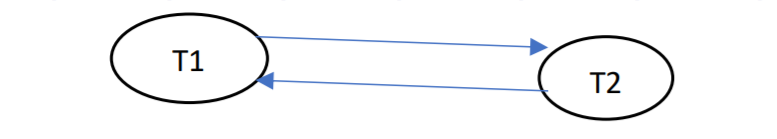
* + Not -Recoverable
  + Not Avoid cascading abort Schedule

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

Ans:- Not Conflict serializable

* + Recoverable
  + Avoid cascading abort Schedule

1. **[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.**
2. read(T1, balx), read(T2, balx), write(T1, balx), write(T2, balx), commit(T1), commit(T2)  
    Ans:-



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



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



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



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

Ans:-

