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.

---

2. **[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	$Total_p$
t1	begin_transaction		0
t2	read($Total_p$)	begin _transaction	0
t3	$Total_p$ += 100	read($Total_p$)	0
t4	write($Total_p$)	$Total_p$ += 50	100

t5	commit	write($Total_p$)	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 $Total_p$ to $100 but it aborts,so $Total_p$ should be back at original value of $0.

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

Time	T3	T4	$Total_p$
t1	begin_transaction		0
t2	read($Total_p$)		0
t3	$Total_p$ += 100		0
t4	write($Total_p$)	begin _transaction	100
t5		read($Total_p$)	100
t6	rollback	$Total_p$ += 50	0
t7		write($Total_p$)	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 $Total_p$ $50 and $Total_p$ $100

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

Time	T5	T6	$Total_p$	$Total_q$	total
t1	begin_transaction		50	100	
t2	total = 0	begin_transaction	50	100	0
t3	read($Total_p$)	read($Total_p$)	50	100	0
t4	total += $Total_p$	$Total_p$ += 50	50	100	50
t5		write($Total_p$)	100	100	50
t6		read($Total_q$)	100	100	50
t7		$Total_q$ -= 50	100	100	50
t8		write($Total_q$)	100	50	50
t9	read($Total_q$)	commit	100	50	50
t10	total += $Total_q$		100	50	100
t11	commit		100	50	100

---

3. **[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	$Total_p$
t1	begin_transaction		0
t2	write_lock($Total_p$)	begin _transaction	0
t3	read($Total_p$)	write_lock($Total_p$)	0
t4	$Total_p$ += 100	**WAIT**	0

t5	write($Total_p$)	**WAIT**	100
t6	commit	**WAIT**	100

t7		read($Total_p$)	100
t8		$Total_p$ += 50	100
t9		write($Total_p$)	150
t10		commit	150

Preventing Uncommitted Dependency Problem using 2PL

Time	T3	T4	$Total_p$
t1	begin_transaction		0
t2	write_lock($Total_p$)		0
t3	read($Total_p$)		0
t4	$Total_p$ += 100	begin _transaction	0
t5	write($Total_p$)	write_lock($Total_p$)	100
t6	rollback	**WAIT**	0
t7		read($Total_p$)	0
t8		$Total_p$ += 50	0
t9		write($Total_p$)	50
t10		commit	50

Preventing Inconsistent Analysis Problem using 2PL

Time	T5	T6	$Total_p$	$Total_q$	total
t1	begin_transaction		50	100	
t2	total = 0	begin_transaction	50	100	0
t3		write_lock($Total_p$)	50	100	0
t4	read_lock($Total_p$)	read($Total_p$)	50	100	0

t5	**WAIT**	$Total_p$ += 50	50	100	0
t6	**WAIT**	write($Total_p$)	100	100	0
t7	**WAIT**	write_lock($Total_q$)	100	100	0
t8	**WAIT**	read($Total_q$)	100	100	0
t9	**WAIT**	$Total_q$ -= 50	100	100	0
t10	**WAIT**	write($Total_q$)	100	50	0
t11	**WAIT**	commit	100	50	0
t11	read($Total_p$)		100	50	0
t11	total += read($Total_p$)		100	50	100
t11	read($Total_q$)		100	50	100
t11	total += $Total_q$		100	50	150
t11	commit		100	50	150

---

4. **[2.5] For each of the following schedules, state whether the schedule is conflict serializable, recoverable and whether it avoids cascading aborts.**

a. read(T_1, bal_x), read(T_2, bal_x), write(T_1, bal_x), write(T_2, bal_x), commit(T_1), commit(T_2)

Ans:- Not -conflict serializable
- Recoverable
- Avoid cascading abort Schedule

b. read(T_1, bal_x), read(T_2, bal_y), write(T_3, bal_x), read(T_2, bal_x), read(T_1, bal_y),commit(T_1), commit(T_2)

Ans:- Conflict serializable
- non-Recoverable
- Not Avoid cascading abort Schedule

c. read(T_1, bal_x), write(T_2, bal_x), write(T_1, bal_x), abort(T_2), commit(T_1)

Ans:- Conflict serializable

- Recoverable
- Avoid cascading abort Schedule

d. write(T_1, bal_x), read(T_2, bal_x), write(T_1, bal_x), commit(T_2), abort(T_1)

Ans:- Conflict serializable
- Not -Recoverable
- Not Avoid cascading abort Schedule

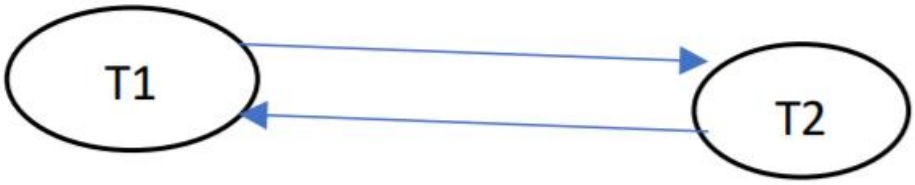
e. read(T_1, bal_x), write(T_2, bal_x), write(T_1, bal_x), read(T_3, bal_x), commit(T_1), commit(T_2), commit(T_3)

Ans:- Not Conflict serializable
- Recoverable
- Avoid cascading abort Schedule

---

5. **[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.**

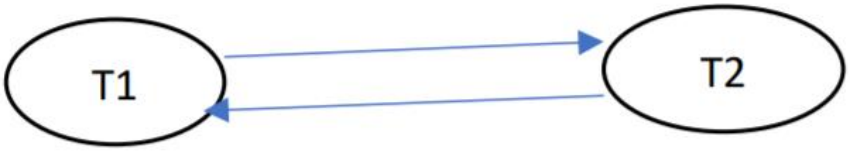
a. read(T_1, bal_x), read(T_2, bal_x), write(T_1, bal_x), write(T_2, bal_x), commit(T_1), commit(T_2)
Ans:-



b. read(T_1, bal_x), read(T_2, bal_y), write(T_3, bal_x), read(T_2, bal_x), read(T_1, bal_y), commit(T_1), commit(T_2)
Ans:-

c. read(T_1, bal_x), write(T_2, bal_x), write(T_1, bal_x), abort(T_2), commit(T_1)
Ans:-

d. write(T_1, bal_x), read(T_2, bal_x), write(T_1, bal_x), commit(T_2), abort(T_1)
Ans:-



e. read(T_1, bal_x), write(T_2, bal_x), write(T_1, bal_x), read(T_3, bal_x), commit(T_1), commit(T_2), commit(T_3)
Ans:-