**DATABASE TRANSACTIONS**

* **Transactions set 1**

Let’s say two transactions

T1: An admin increases the stock quantity of a product A by 100 lets say and B by 300.

T2: An admin increases the stock quantity of a product A by 200 lets say.

Now we can build schedules that may either be conflicting serializable or non conflicting serializable.

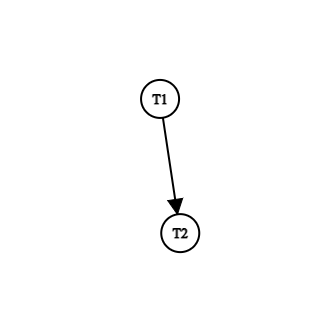
SCHEDULE 1) CONFLICTING SERIALIZABLE

| T1 | T2 |
| --- | --- |
| R(A):Take input of stock quantity of product A |  |
| A=A+100; |  |
| W(A): Update in database |  |
|  | R(A):Take input of stock quantity of product A |
|  | A=A+200 |
|  | W(A): Update in database |
|  | commit; |
| R(B):Take input of stock quantity of product B |  |
| B=B+300 |  |
| W(B): Update in database |  |
| commit; |  |

This type of conflict includes a WR type of conflict the T2 dirty reads from the T1.

This is conflict serializable because:

If we draw T1 and T2 as nodes we see a conflict WR occurs as underlined thus an arrow from T1 TO T2 as the graph so formed is A DAG therefore it is a conflict serializable transaction schedule.



**PRECEDENCE GRAPH**

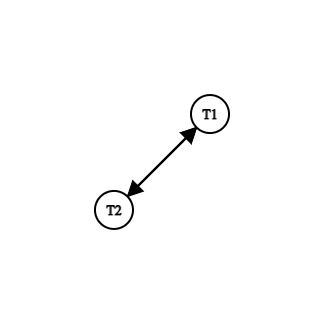
SCHEDULE 2) NON CONFLICTING SERIALIZABLE:

| T1 | T2 |
| --- | --- |
|  | R(A): Take input of stock quantity of A |
|  | A=A+300; |
| R(A): Take input of stock quantity of A |  |
| A=A+100; |  |
| W(A): Update in database |  |
|  | W(A): Update in database |
|  | Commit; |
| R(B):Take input of stock quantity of product B |  |
| B=B+300 |  |
| W(B): Update in database |  |
| commit; |  |

In this RW and WW type of conflict takes place for product A.

This is non-conflict serializable schedule because:

When we make the precedence graph for the two transactions a loop is obtained in the graph hence this is a non-conflict serializable transaction whose transactions can’t be swapped to obtain serializable scheduling.



THERE IS A RW CONFLICT FROM T2 TO T1 AND AN RW AND A WW CONFLICT FROM T1 TO T2 HENCE A LOOP IS OBTAINED .

Non-conflict serializability solved using Strict two phase locking:

S(A):Shared lock on A for read only

U(A):unlock on A

X(A):exclusive lock on A for read and write both;

| T1 | T2 |
| --- | --- |
|  | S(A) |
|  | R(A): Take input of stock quantity of A |
|  | A=A+300; |
| X(A) |  |
| R(A): Take input of stock quantity of A |  |
| A=A+100; |  |
| W(A): Update in database |  |
|  | W(A): Update in database |
|  | Commit; |
|  | U(A) |
| X(B) |  |
| R(B):Take input of stock quantity of product B |  |
| B=B+300 |  |
| W(B): Update in database |  |
| commit; |  |
| U(A) |  |
| U(B) |  |

* **Transaction set 2:**

**Lets take the example of two delivery partners selecting some orders;**

T1: DP1 selects order A and order B;

T2: DP2 selects order A;

Let R(A) means reading input of order A

And W(A) means updating their name in front of an unselected order A to select it.

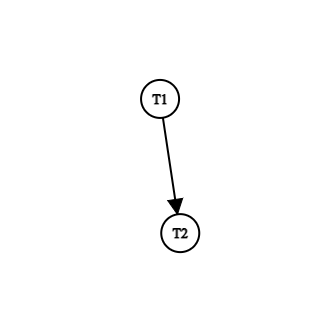
If the order is already selected then don’t update .

SCHEDULING 1) CONFLICTING SERIALIZABLE SCHEDULE:

| T1 | T2 |
| --- | --- |
| R(A) |  |
| W(A) |  |
|  | R(A) |
|  | W(A) |
|  | COMMIT; |
| R(B) |  |
| W(B) |  |
| COMMIT; |  |

Here it has WR conflict as highlighted .What happens lets say DP1 accepts order A and update its name then when the T2 takes place DP2 won’t be able to accept the order as it is already accepted if before T1 commits it fails then the whole transaction is aborted and it roll back.Then both of them didn't get to accept order A thus huge amount of transaction is wasted.

This is conflict serializable because for nodes T1 and T2 there is an arrow from T1 to T2 for WR conflict and thus the precedence graph is a DAG hence a conflict serializable.



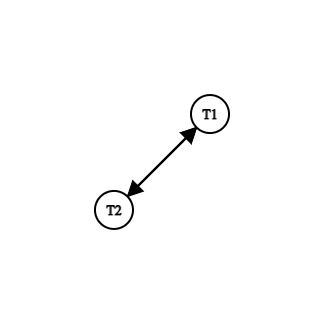
SCHEDULING 2: NON-CONFLICT SERIALIZABLE:

| T1 | T2 |
| --- | --- |
|  | R(A) |
| R(A) |  |
| W(A) |  |
|  | W(A) |
|  | COMMIT; |
| R(B) |  |
| W(B) |  |
| COMMIT; |  |

Here it has WW conflict as highlighted.

Lets say DP2 reads that order A is unselected and before it updates DP1 reads the order A and updates its name now when the DP2 tries to update error is generated.

This is a nonconflicting serializable schedule because in precedence graph there is a loop in T1 and T2 hence it is not DAG hence not conflicting serializable.For two nodes T1 AND T2 there is an arrow from T2 to T1 for RW and there is an arrow from T1 to T2 for RW and WW.



Non-conflict serializability solved using Strict two phase locking:

S(A):Shared lock on A for read only

U(A):unlock on A

X(A):exclusive lock on A for read and write both;

| T1 | T2 |
| --- | --- |
|  | S(A) |
|  | R(A) |
| X(A) |  |
| R(A) |  |
| W(A) |  |
|  | X(A) |
|  | W(A) |
|  | COMMIT; |
|  | U(A) |
| X(B) |  |
| R(B) |  |
| W(B) |  |
| COMMIT; |  |
| U(A) |  |
| U(B) |  |