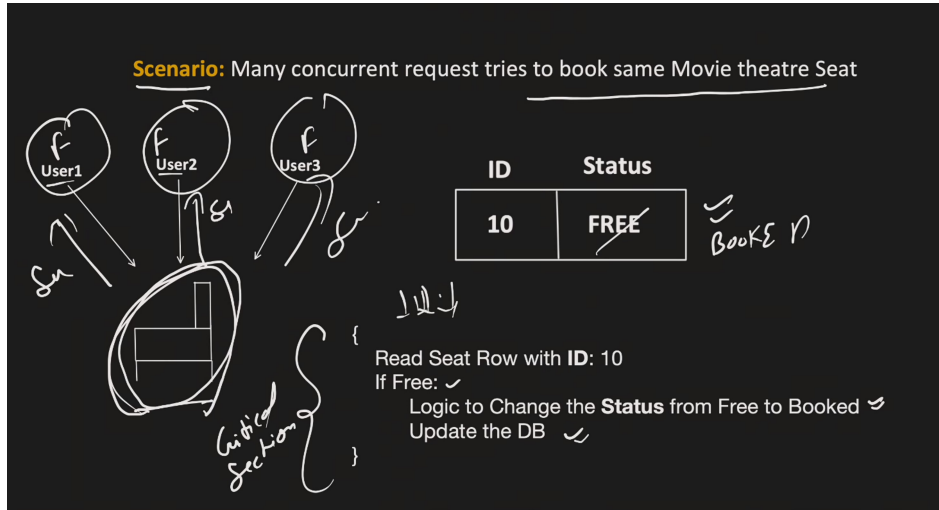


Distributed Concurrency Control

Wednesday, October 18, 2023 3:20 PM

This will help us to understand how to handle multiple concurrent request.

Many Concurrent request tries to book same Movie Theatre Seat.



Use Synchronized () for the critical Section:

```
Synchronized()  
{  
  Read Seat Row with ID : 10  
  If Free :  
    Logic to change the status from FREE to Booked.  
    Update the DB.  
}
```

But this solution will not work for distributed system, as we have different process, and Synchronized will work on Threads, not on process.

NOW we will jump to **Distributed Concurrency Control**

What is the usage of a Transaction?

- Transaction Helps to achieve **INTEGRITY**. Means it help us to avoid **INCONSISTENCY** in our DATABASE.

Transaction helps to achieve **INTEGRITY**. Means it help us to avoid **INCONSISTENCY** in our database.

For example: ✓

Debit the Money from A and Credit the money to B

Dr A
Cr B

A: 100
B: 50

BEGIN_TRANSACTION:

- Debit the Money from A *200*
- Credit the Money to B

If all success:

COMMIT;

else:

ROLLBACK;

END_TRANSACTION;

What is DB Locking ?

- DB Locking, help us to make sure that no other transaction update the locked rows.
- Shared lock / Read Lock -- Only read can happen, Multiple Transaction can come into picture and can read the DB.
- Exclusive lock - if one transaction has taken on Exclusive lock then no other transaction can take shared or exclusive lock.

LockType	Another Shared Lock	Another Exclusive Lock
Have Shared Lock	Yes	No
Have Exclusive Lock	No	No

What are the Isolation Level Present ?

Isolation means, whenever transactions works it feels like they are running alone apart from actual scenario where there are multiple Transactions running in parallel.

3. What are the Isolation Level present?

Isolation Level	Dirty Read Possible	Non-Repeatable Read Possible	Phantom Read Possible
① Read Uncommitted	Yes	Yes	Yes
② Read Committed	No	Yes	Yes
③ Repeatable Read	No	No	Yes
④ Serializable	No	No	No

Consistency High

Consistency Low

Dirty Read :

READ UNCOMMITTED : Use it only when READ is required.

	ISOLATION LEVEL	Locking Strategy
(0)	Read Uncommitted	Read : No Lock acquired Write : No Lock acquired
(1)	Read Committed	Read : Shared Lock acquired and Released as soon as Read is done Write : Exclusive Lock acquired and keep till the end of the transaction
(2)	Repeatable Read	Read : Shared Lock acquired and Released only at the end of the Transaction Write : Exclusive Lock acquired and Released only at the end of the Transaction
(3)	Serializable	Same as Repeatable Read Locking Strategy + apply Range Lock and lock is release only at the end of the Transaction.

Dirty Read :

If Transaction A is reading the data which is writing by Transaction B and not yet even committed.

If Transaction B does the rollback, then whatever data read by Transaction A is known Dirty Read.

Time	Transaction A	Transaction B	DB
T1	BEGIN_TRANSACTION	BEGIN_TRANSACTION	ID: 1 Status: Free
T2	Update Row ID:1, Status: Booked		ID: 1 Status: Booked (Not Committed by Transaction B Yet)
T3		Read Row ID:1 (Got status as Booked)	ID: 1 Status: Booked (Not Committed by Transaction B Yet)
T4	Some Issue faced here	Some Computation	ID: 1 Status: Booked (Not Committed by Transaction B Yet)
T5	Rollback	Commit	ID: 1 Status: Free

Non - Repeatable Read:

If suppose Transaction A, reads the same row several times, and there is a chance that it reads different value.

TS

	Transaction A	DB
<u>T1</u>	BEGIN_TRANSACTION	ID: 1 Status: Free
T2	Read Row ID:1 (reads status: Free)	ID: 1 Status: Free
T3		ID: 1 Status: Booked ← Some other Transaction changed and committed the changes.
T4	Read Row ID:1 (reads status: Booked)	ID: 1 Status: Booked
T5	COMMIT	

Phantom Read :

If Suppose Transaction A, executes same query several times and there is a chance that the rows returned are different.

	Transaction A	DB
T1	BEGIN_TRANSACTION	<div>ID: 1, Status: Free</div> <div>ID: 3, Status: Booked</div>
T2	Read Row where ID>0 and ID<5 (reads 2 rows ID:1 and ID:3)	<div>ID: 1, Status: Free</div> <div>ID: 3, Status: Booked</div>
T3		<div>ID: 1, Status: Free</div> <div>ID:2, Status: Free</div> <div>ID: 3, Status: Booked</div>
T4	Read Row where ID>0 and ID<5 (reads 3 rows ID:1, ID:2 and ID:3)	<div>ID: 1, Status: Free</div> <div>ID:2, Status: Free</div> <div>ID: 3, Status: Booked</div>
T5	COMMIT	

Some other Transaction Inserted the row with ID:2 and Committed

Optimistic Concurrency Control (OCC):

- Isolation Level used Below Repeatable Read
- Much Higher Concurrency
- No change of DeadLock
- In Case of conflict, overhead of transaction rollback and retry logic is there.

Pessimistic Concurrency Control (PCC)

- Isolation Level used REPEATABLE READ and SERIALIZABLE
- Less Concurrency compared to Optimistic
- Deadlock is possible, then transactions stuck in deadlock forced to do rollback.
- Putting a long lock, sometimes timeout issue comes and rollback need to be done.

Optimistic Concurrency Control (OCC) :

Rea

	Transaction A	Transaction B	DB
T1	BEGIN_TRANSACTION	BEGIN_TRANSACTION	ID: 1 Status: Free Version: 1
T2	Read Row ID:1 (Row Version is 1)	Read Row ID:1 (Row Version is 1)	ID: 1 Status: Free Version: 1
T3	Select for Update (Version Validation happens)		ID: 1 Status: Free Version: 1 (Exclusive Lock by Txn A)

T4	Update Row ID:1, Status: Booked, Version:2		ID: 1 Status: Booked (Exclusive Lock by Txn A) Version:2
T5	COMMIT		ID: 1 Status: Booked Version:2
T6		Select for Update (Version Validation happens)	ID: 1 Status: Booked (Exclusive Lock by Txn B) Version:2