Distributed Concurrency Control

Dirty Read:
If Transaction A is reading the data which is written by Transaction B and not yet committed.
If Transaction B does the rollback then whatever data read by Transaction A is known as dirty read.

Non-Repeatable Read:
If a transaction reads a row several times but reads different values because some other transaction has changed it in the middle.

Phantom Read:
If suppose a transaction executes the same query several times and the rows returned are different.

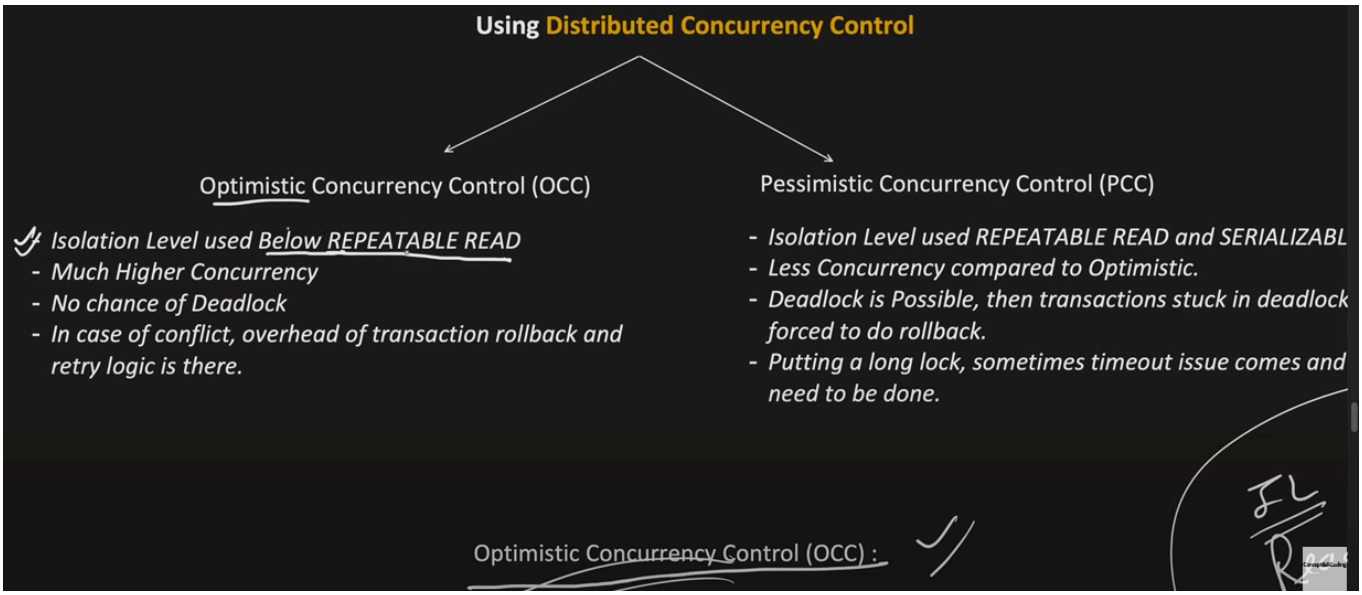
Isolation Level	Dirty Read Possible	Non-Repeatable Read Possible	Phantom Read Possible
Read Uncommitted	Yes	Yes	Yes
Read Committed	No	Yes	Yes
Repeatable Read	No	No	Yes
Serializable	No	No	No

Consistency High

Consistency Low

ISOLATION LEVEL	Locking Strategy
Read Uncommitted	**Read** : No Lock acquired **Write**: No Lock acquired
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
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
Serializable	Same as Repeatable Read Locking Strategy + apply Range Lock and lock is release only at the end of the Transaction.

Consistency=>Concurrency



Key Characteristics of Pessimistic Locking:

1. **Acquiring Locks Upfront:** Before performing any read or write operation on a resource, pessimistic locking acquires locks on the resource to prevent other concurrent transactions from accessing it concurrently.
2. **Exclusive Access:** Once a lock is obtained, it grants exclusive access to the locked resource to the transaction that holds the lock. Other transactions must wait until the lock is released before they can access the resource.
3. **Potential for Waiting:** Since locks are held for the entire duration of a transaction, there might be waiting times for other transactions if they request access to the locked resource.
4. **Different Granularities:** Pessimistic locking can operate at various granularities, such as database-level locks, table-level locks, row-level locks, or even finer-grained locks at the column level.

✅ Pros

- Ensures data consistency by preventing concurrent modifications that could lead to inconsistencies.
- Straightforward approach to managing concurrent access.

❌ Cons

- Potential for increased contention and waiting times, especially in scenarios with high concurrency.
- Locks held for extended periods might impact system performance and throughput.

When to use Pessimistic Locking

- **High Contention:** In scenarios where conflicts are frequent or expected, such as in banking systems handling critical transactions, pessimistic locking might be more suitable. It ensures exclusive access to resources, reducing the chance of conflicts and maintaining data integrity.
- **Long Transactions:** When transactions hold locks for extended periods due to complex operations or user interactions, pessimistic locking can prevent inconsistencies by keeping resources locked until the transaction completes.

Key Concepts of Optimistic Locking

1. **Validation Before Commit:** Rather than preventing concurrent access, optimistic locking defers conflict detection until the time of committing changes. It allows transactions to proceed independently and optimistically assumes that conflicts won't arise during the execution phase.
2. **Validation at Commit Time:** Before committing changes, the system checks whether any other transaction has modified the resource since the current transaction started. This validation is typically done by comparing versions or timestamps associated with the resource.
3. **Handling Conflicts:** If conflicts are detected during the validation phase (e.g., if another transaction modified the resource), the system takes appropriate action. This could involve rolling back the current transaction or retrying it to incorporate the latest changes.

✅ Pros

- Reduced contention: Allows concurrent access without blocking, potentially improving system concurrency.
- Minimal locking overhead: Transactions proceed independently, reducing the need for acquiring and managing locks.

❌ Cons

- Increased chances of conflicts: Might require retrying transactions or handling conflicts, leading to additional logic and complexity.
- Not suitable for highly contended resources: In scenarios with frequent conflicts, optimistic locking might not be the most efficient approach.

When to use Optimistic Locking

- **Low Contention:** In scenarios where conflicts are rare or infrequent, such as in read-heavy systems or non-critical data operations, optimistic locking can be more suitable. It allows concurrent access without blocking, potentially improving system concurrency.

- **Short Transactions:** When transactions are short-lived and the likelihood of conflicts is minimal, optimistic locking’s non-blocking nature can be beneficial, as it avoids unnecessary waiting times.