

# Isolation and Its Types

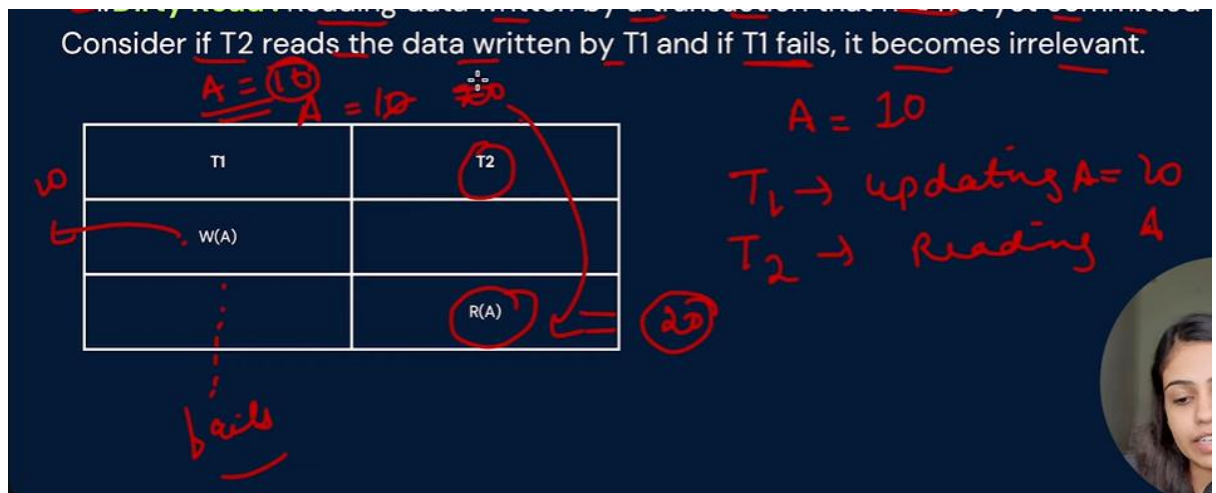
- **Why do we need isolation level?**

In systems where multiple transactions are executed concurrently, isolation levels manage the extent to which the operations (read, write) of one transaction are isolated from those of other transactions.

It determines the degree to which the operations in one transaction are isolated from those in other transactions.

Isolation levels help prevent common transactional anomalies:

1. **Dirty Read:** Reading uncommitted data from another transaction. (if T1 comes and update a value to 20 from 10, but it does not commit, and T2 read that 20 value and then T1 fails and rollback to 10 again but value 20 read by T2 is invalid)



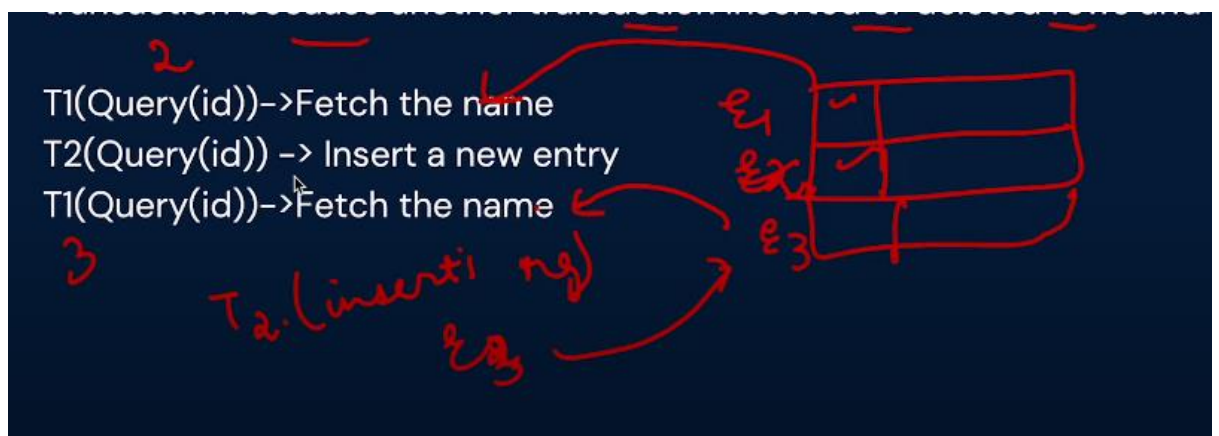
2. **Non-repeatable Read:** Reading the same row twice within a transaction and getting different values because another transaction modified the row and committed.

Consider if T2 modifies the data which T1 already Read and if T1 continue the transaction the data will be changed



- Phantom Read:** New rows are added or removed by another transaction after a query. (ie. if before, one query comes and reads the no. of rows be 2, and then one row is added then, the read by the previous query read/result is invalid or incorrect or inconsistent as now there is 3 rows).

Getting different sets of rows in subsequent queries within the same transaction because another transaction inserted or deleted rows and committed.



- Type of Isolation levels

1. Read Uncommitted:

**Read Uncommitted:** The lowest isolation level where transactions can see uncommitted changes made by other transactions. If Transaction T1 is writing a value to a table, Transaction T2 can read this value before T1 commits.

- Dirty Reads: Yes
- Non-Repeatable Reads: Yes
- Phantom Reads: Yes

$T_1 \rightarrow W(A) \leftarrow R(A) \leftarrow T_2$

2. Read Committed:

**Read Committed:** It ensures that any data read during the transaction is committed at the moment it is read. If T1 has done some write operation T2 can only read the data when T1 is committed.

- Dirty Reads: No
- Non-Repeatable Reads: Yes
- Phantom Reads: Yes

$T_1 \rightarrow W(A) \leftarrow R(A) \leftarrow T_2$

Non-repeatable reads present as, T1 read a value, then T2 comes and change the value (commits), so next time T1 reads, it gets different value. Like this phantom reads also happen.

3. Repeatable Read:

It ensures that if a transaction reads a row, it will see the same values for that row during the entire transaction, even if other transactions modify the data and commit. If Transaction T1 reads a value, Transaction T2 cannot modify that value until T1

completes. But T2 can insert new rows that T1 can see on subsequent reads (phantom reads).

### **Timeline of Events: (Row Level Blocking)**

1. **Transaction 1** starts and reads Account\_ID 102 (balance = 1000).
2. **Transaction 1** locks the row for Account\_ID 102.
3. **Transaction 2** attempts to update Account\_ID 102 (new balance = 1200) but is **blocked** because the row is locked by **Transaction 1**.
4. **Transaction 1** completes (either by committing or rolling back).
5. The lock is released.
6. **Transaction 2** is unblocked and proceeds with updating the balance to 1200.

### **Timeline of Events: (Multi-Version Concurrency Control MVCC)**

1. T1 starts a transaction and reads the balance of Account\_ID 102, which is 1000.
2. T2 starts a new transaction and updates the balance of Account\_ID 102 to 1200. Under MVCC:
3. The system creates a new version of the row for Account\_ID 102 with the updated balance of 1200.
4. T2 works with this new version.
5. T1 continues to see the old version of the row with a balance of 1000, because that was the state when it started.
6. T1 commits, but the balance it sees during its transaction is always 1000.
7. T2 commits, and now the balance for Account\_ID 102 is officially 1200.

\* Dirty Reads: No

\* Non-Repeatable Reads: No

\* Phantom Reads: Yes

#### 4. Serializable:


**LET'S START WITH DBMS :)**

Isolation levels and its types

↓

**Serializable:** It ensures a serial transaction execution, so that there is complete isolation. If Transaction T1 is executing, Transaction T2 must wait until T1 completes

- Dirty Reads: No
- Non-Repeatable Reads: No
- Phantom Reads: No



Serializable has the most extent of isolation.