|  |  |
| --- | --- |
| Propagation | Behavior |
| **REQUIRED** | **Always executes in a transaction.** If there is any existing transaction it uses it. If none exists then only a new one is created |
| **SUPPORTS** | **It may or may not run in a transaction.** If current transaction exists then it is supported. If none exists then gets executed without transaction. |
| **NOT\_SUPPORTED** | **Always executes without a transaction**. If there is any existing transaction it gets suspended |
| **REQUIRES\_NEW** | **Always executes in a new transaction.** If there is any existing transaction it gets suspended |
| **NEVER** | **Always executes without any transaction**. It throws an exception if there is an existing transaction |
| **MANDATORY** | **Always executes in a transaction**. If there is any existing transaction it is used. If there is no existing transaction it will throw an exception. |

**What is Transaction Propagation?**  
Any application involves a number of services or components making a call to other services or components. Transaction Propagation indicates if any component or service will or will not participate in transaction and how will it behave if the calling component/service already has or does not have a transaction created already.

**What is Transaction Isolation?**  
Transaction Isolation defines the database state when two transactions concurrently act on the same database entity. It involves locking of database records. So it describes the behavior or state of the database when one transaction is working on database entity and then some other concurrent transaction tries to simultaneously access/edit the same database entity.

**SERIALIZABLE:** If two transactions are executing concurrently then it is as if the transactions get executed serially i.e the first transaction gets committed only then the second transaction gets executed. This is total isolation. So a running transaction is never affected by other transactions. However this may cause issues as performance will be low and deadlock might occur.

**REPEATABLE\_READ:** If two transactions are executing concurrently - till the first transaction is committed the existing records cannot be changed by second transaction but new records can be added. After the second transaction is committed, the new added records get reflected in first transaction which is still not committed.

**READ\_COMMITTED:** If two transactions are executing concurrently - before the first transaction is committed the existing records can be changed as well as new records can be changed by second transaction. After the second transaction is committed, the newly added and also updated records get reflected in first transaction which is still not committed.

**READ\_UNCOMMITTED:** If two transactions are executing concurrently - before the first transaction is committed the existing records can be changed as well as new records can be changed by second transaction. Even if the second transaction is not committed the newly added and also updated records get reflected in first transaction which is still not committed.