**Transaction**

It is unit of work performed within a DBMS

It comes with begin and commit.

**Transactional boundary (limit, domain)**

In java a method that has @Transactional or in database begin-commit block

**Transactional Database**

if [DBMS](https://en.wikipedia.org/wiki/DBMS)  provides the [ACID properties](https://en.wikipedia.org/wiki/ACID_properties) for a database operations from within 1 block of (begin-commit).

**ACID properties in database**

* **Atomic** 
  + Guarantee that all the operations from within begin-commit treated as a single unit
* **Consistency**
  + Guarantee that a data written are valid as per rules such as constraints, triggers, cascades
* **Isolation**
  + Ensures that concurrency is controlled , the effects of an incomplete transaction might not be visible to other transactionsgg
* **Durability** 
  + guarantees once transaction has been committed it will remain committed even in the case of system failures

**Transaction management**

Mechanism to handle performing distributed operations 1 block begin-commit or

Handling propagation c

**Transaction Propagation**

Describes how the transactions could be related to each other as per business requirement

Developers can decide how to encapsulate transactions in both physical and logical transactions

**Physical Transaction**

Refers to actual database or JDBC transaction that starts with begin-commit block of the code

**Logical Transaction**

Refers to transaction that provides by JPA and Hibernate and often takes from dataSource.

**Transaction Propagation type**

[**https://en.wikibooks.org/wiki/Java\_Persistence/Transactions**](https://en.wikibooks.org/wiki/Java_Persistence/Transactions)

**REQUIRED** : Reuse the same transaction if there is any or create new one

**REQUIRED**\_**NEW**: Code will run the current transaction and new will be created.

**Distinct Logical Transaction**

If multiple methods configured as **required** 🡪

* They will be shared the same Distinct logical transaction
* The will be in the same physical transaction
* In nested call failure in one will rollback the rest

If method A(**REQUIRED**) is called if there is no TX then 1 will be created otherwise reuse the same

If method A called methodB(**REQUIRED\_NEW**) , TX1 will be paused , new physical Transaction will be created inside method B. The result of method B transation does not reflect Transaction in method A  
They are both in different transaction

**NOT\_SUPPORTED**

No TX started. Throws exception if attempts UPDATE/SAVE and will rollback the first TX

**SUPPORTED**

No required TX . Throws exception if attempts UPDATE/SAVE and will rollback the first TX   
If called by a method that has TX then will be as part of it

**Mandatory**

Required open tx. If not throws an exception

**Isolation**

Ensures that concurrency is controlled , An incomplete transaction might not be visible to other transactions

**Isolation level**

A degree in which a data that has modified is visible to other transactions

**Transaction lcoking**

Supports an fulfill isolation level in Transaction

**Locking mode**

* **Read un-committed**: Allows dirty read
* **Read committed**: does not allow dirty read
* **Repeatable read**: if rows reads twice in the same TX then result is always the same
* **Serializable** : Performs transactions in sequences

No other transactions can modify data that has been read by the current transaction until the current transaction completes.

**Default lock Spring**

It will be a database default locking , eg

Oracle is Read-committed 🡪 second TX does not see changes by first TX   
SQL server is Read-commited

Avoid dead lock in Transaction

1. Use read-committed lock to ensure no lock occurs
2. Make the transaction as short as possible
3. Avoid cursor
4. Reduce lock time