**Database Anomalies – Quick Reference – 2022**

In database systems, isolation determines how transaction integrity is visible to other users and systems.

**Dirty reads**

A *dirty read* (aka *uncommitted dependency*) occurs when a transaction is allowed to read data from a row

that has been modified by another running transaction and not yet committed.

Graphical user interface, text, application, chat or text message

Description automatically generated

READ\_UNCOMMITTED

READ\_UNCOMMITTED isolation level states that a transaction may read data that is still uncommitted by other transactions. This constraint is very relaxed in what matters to transactional concurrency but it may lead to some issues like dirty reads.

READ\_COMMITTED

**READ\_COMMITTED** isolation level states that a transaction can't read data that is **not** yet committed by other transactions. This means that the **dirty read** is no longer an issue

**@Transactional(isolation = Isolation.READ\_COMMITTED)**

**public** **void** **log**(String message){

// ...

}

**READ\_COMMITTED is the default level with Postgres, SQL Server, and Oracle.**

### **Non-repeatable reads**

A *non-repeatable read* occurs when, during the course of a transaction, a row is retrieved twice and the values within the row differ between reads.

Graphical user interface, text, application, email

Description automatically generated

## REPEATABLE\_READ

**REPEATABLE\_READ** isolation level states that if a transaction reads one record from the database multiple times the result of all those reading operations must always be the same. This eliminates both the **dirty read** and the **non-repeatable read** issues.

**@Transactional(isolation = Isolation.REPEATABLE\_READ)**

**public** **void** **log**(String message){

// ...

}

**REPEATABLE\_READ is the default level in Mysql. Oracle does not support REPEATABLE\_READ.**

### **Phantom reads**

A *phantom read* occurs when, in the course of a transaction, new rows are added or removed by another transaction to the records being read.

Graphical user interface, text, application, chat or text message

Description automatically generated

## SERIALIZABLE

**SERIALIZABLE** isolation level is the most restrictive of all isolation levels. Transactions are executed with locking at all levels. This solves Dirty Reads, NonRepeatable Read and Phantom Read.

**@Transactional(isolation = Isolation.SERIALIZABLE)**

**public** **void** **log**(String message){

// ...

}

Each isolation level prevents zero or more concurrency side effects on a transaction:

* **Dirty read:** read the uncommitted change of a concurrent transaction
* **Nonrepeatable read**: get different value on re-read of a row if a concurrent transaction updates the same row and commits
* **Phantom read:** get different rows after re-execution of a range query if another transaction adds or removes some rows in the range and commits

According to the [docs (Isolation.DEFAULT)](http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/transaction/annotation/Isolation.html#DEFAULT), it uses

Use the default isolation level of the underlying datastore.

As you are using the @Transactional annotation, I would set the isolation level there, e.g.:

@Transactional(propagation=Propagation.REQUIRES\_NEW, isolation=Isolation.SERIALIZABLE)

The default isolation level is DEFAULT. As a result, when Spring creates a new transaction, the isolation level will be the default isolation of our RDBMS. Therefore, we should be careful if we change the database