Transactions

WCS: SEA Java Telekom

Agenda

- What is a transaction
- Begin, Commit, Rollback
- Isolation Level
- Transactions with JDBC, JPA
- Spring @Transactional
- Transaction Propagation
- 2 Phase Commit

Transaction = Logical Unit of Work

- All steps have to complete successfully (Commit)
- If one step fails all executed steps should be reversed (Rollback)

Example: Bank Transfer

- Substract 100 EUR of one bank account
- Add 100 EUR to another account

If one step fails there would be money lost or created

Why do I need to know?

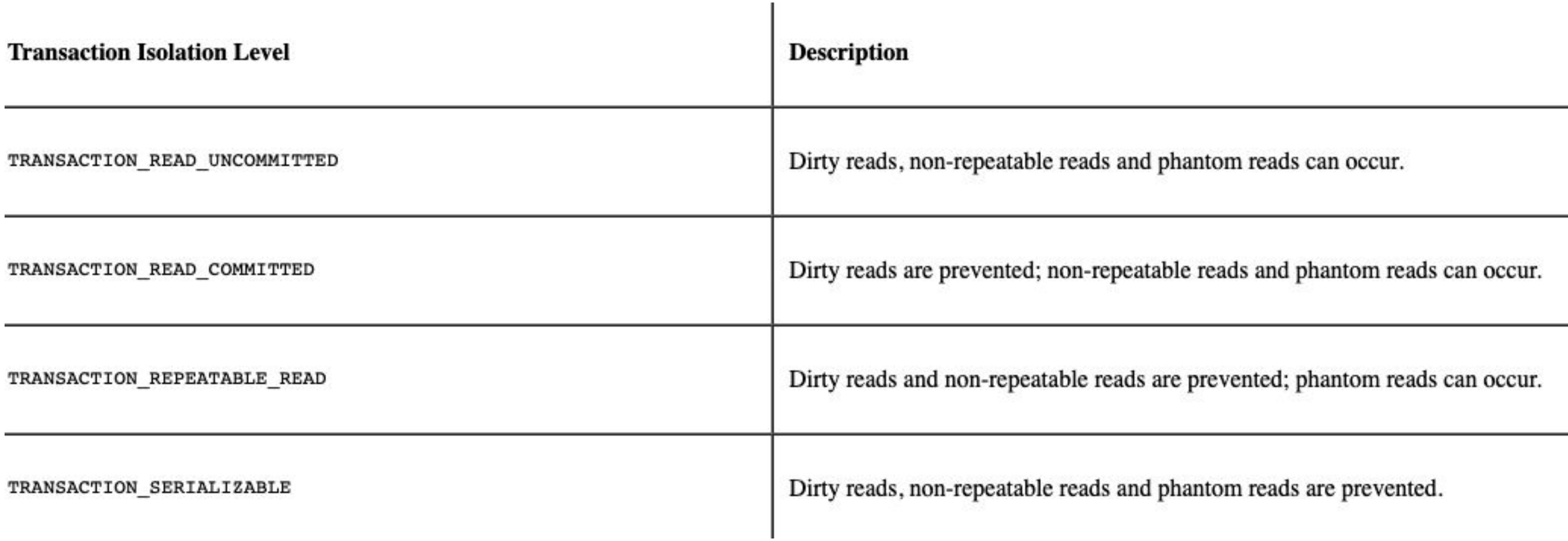
Properties of a transaction

- A - Atomic (As one unit)
- C - Consistent (Example Banktransfer)
- I - Isolated (Other transfers may not interfere)
- D - Durable (Changes are persisted)

Konto1 (Guthaben 100): Abheben -100 (

Konto2: Gutschreiben +100

Isolation Level



Transaction Isolation Level	Description
`TRANSACTION_READ_UNCOMMITTED`	Dirty reads, non-repeatable reads and phantom reads can occur.
`TRANSACTION_READ_COMMITTED`	Dirty reads are prevented; non-repeatable reads and phantom reads can occur.
`TRANSACTION_REPEATABLE_READ`	Dirty reads and non-repeatable reads are prevented; phantom reads can occur.
`TRANSACTION_SERIALIZABLE`	Dirty reads, non-repeatable reads and phantom reads are prevented.

Transactions with JDBC

```
Class.forName("com.mysql.jdbc.Driver");
Connection con =
DriverManager.getConnection("jdbc:mysql://my_stuff:my_stuff@localhost:3306/my_stuff?serverTimezone=CET");

try {
	con.setAutoCommit(false);
	// con.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);
	Statement st = con.createStatement();
	st.execute("UPDATE BANKACCOUNTS SET AMOUNT=1000 WHERE USER='david'");
	st.execute("UPDATE BANKACCOUNTS SET AMOUNT=2000 WHERE USER='andre'");
	con.commit();
} catch(Exception ex) {
	con.rollback();
}finally {
	con.close();
}
```

Quest

Create a transfer Service that can withdraw from one account and deposit to another account in one transaction.

Use a standalone java program with jdbc, mysql

https://github.com/beisdog/wcs-java-transactions-quest.git

wcs-java-tx-jdbc-simple-quest/

Quest Continued

Create table:

```
CREATE TABLE bankaccounts (
	id INT NOT NULL AUTO_INCREMENT,
	user VARCHAR(45) NOT NULL,
	balance DECIMAL(12,2) NOT NULL DEFAULT 0,
	PRIMARY KEY (id)
);
```

Transactions with Plain JPA

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("Bank");
EntityManager em = emf.createEntityManager();
EntityTransaction tx = em.getTransaction();

tx.begin();
BankAccount toAccount = (BankAccount) this.em.createQuery("FROM BankAccount b where b.user =
andre").getSingleResult();

toAccount.setBalance(toAccount.getBalance().add(amount));
BankAccount fromAccount = (BankAccount) this.em.createQuery("FROM BankAccount b where b.user =
‘david’").getSingleResult();

fromAccount.setBalance(fromAccount.getBalance().substract(amount));
tx.commit();
em.close();
```

Quest

Create a transfer Service that can withdraw from one account and deposit to another account in one transaction.

Use a standalone java program with JPA

https://github.com/beisdog/wcs-java-transactions-quest.git

wcs-java-tx-jpa-simple-quest/

Transactions with Spring and @Transactional

```
@Service
public class TransferService {

    @Autowired
    private AccountService service;

    @Transactional(value = TxType.REQUIRED)
    public List<BankAccount> transferMoney(String userFrom, String userTo, BigDecimal amount)
            throws InsufficientFundsException {
        BankAccount toAccount = service.deposit(userTo, amount);
        BankAccount fromAccount = service.withdraw(userFrom, amount);
        return Arrays.asList(fromAccount, toAccount);
    }
}
```

Exception Handling

- Default behaviour
    - Rollback for unchecked exception
    - No rollback for checked exception
- Can be changed with @Transactional properties
    - rollbackOn

Transactions @Transactional(rollbackOn=...)

```
@Service
public class TransferService {

    @Autowired
    private AccountService service;

    @Transactional(value = TxType.REQUIRED, rollbackOn = InsufficientFundsException.class)
    public List<BankAccount> transferMoney(String userFrom, String userTo, BigDecimal amount)
            throws InsufficientFundsException {
        BankAccount toAccount = service.deposit(userTo, amount);
        BankAccount fromAccount = service.withdraw(userFrom, amount);
        return Arrays.asList(fromAccount, toAccount);
    }
}
```

Transaction Propagation

NOT_SUPPORTED

NEVER

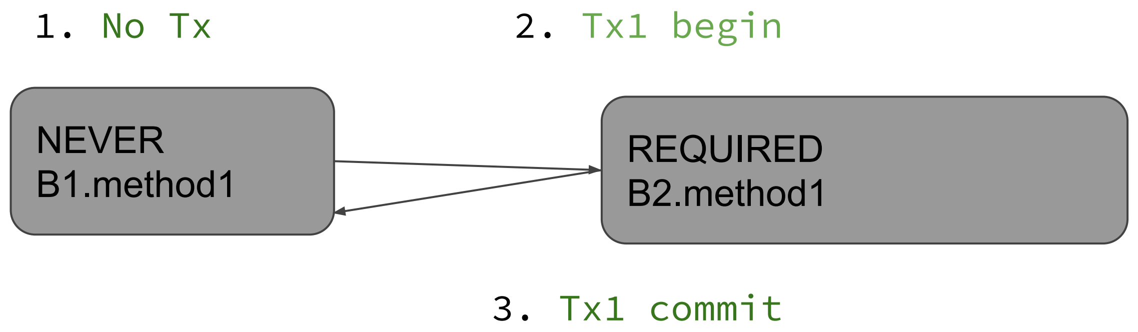
SUPPORTS

REQUIRED

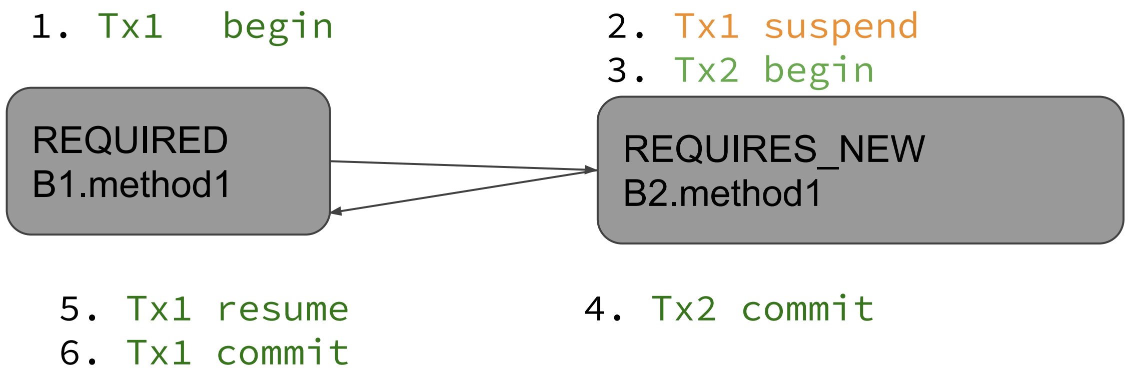
MANDATORY

REQUIRES_NEW

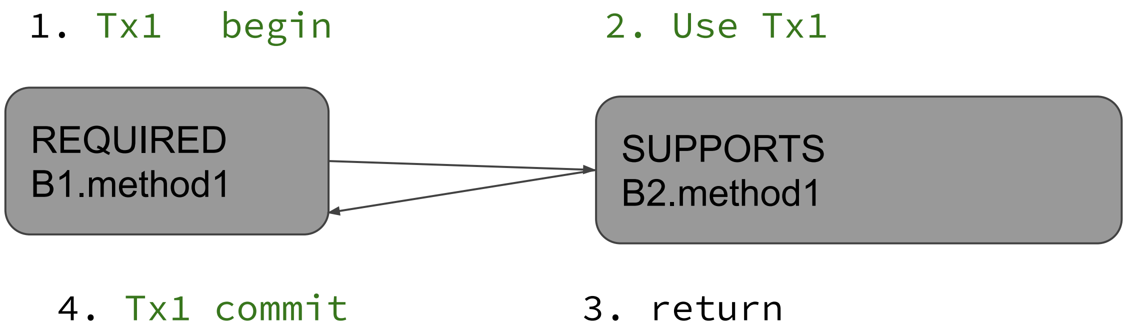
How propagation works: REQUIRED, NEVER

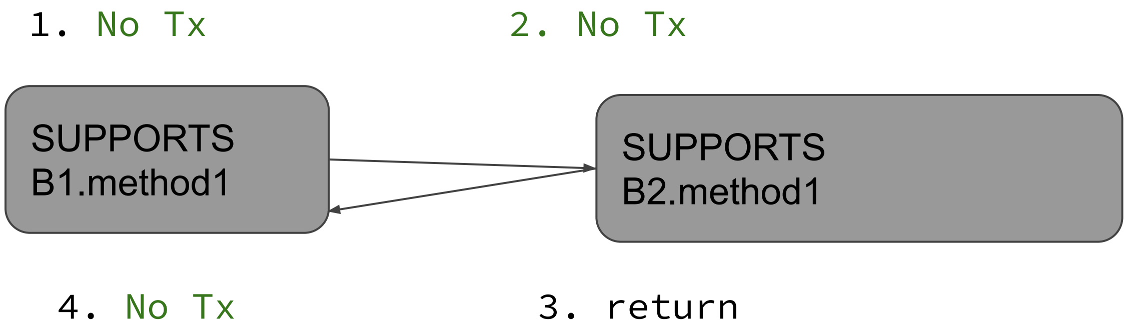


How propagation works: REQUIRES_NEW

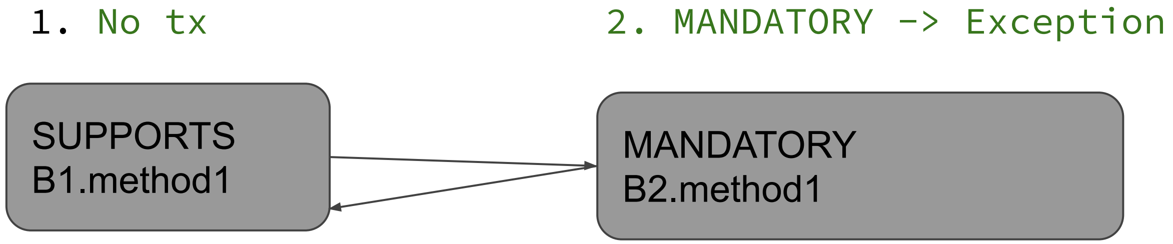


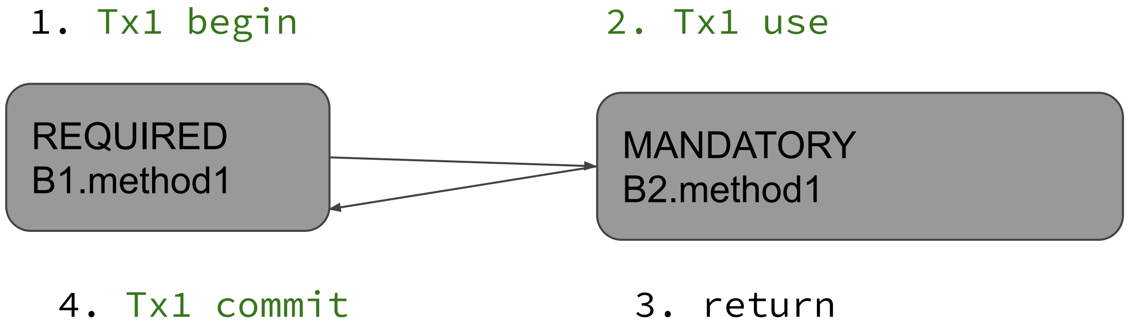
How propagation works: SUPPORTS



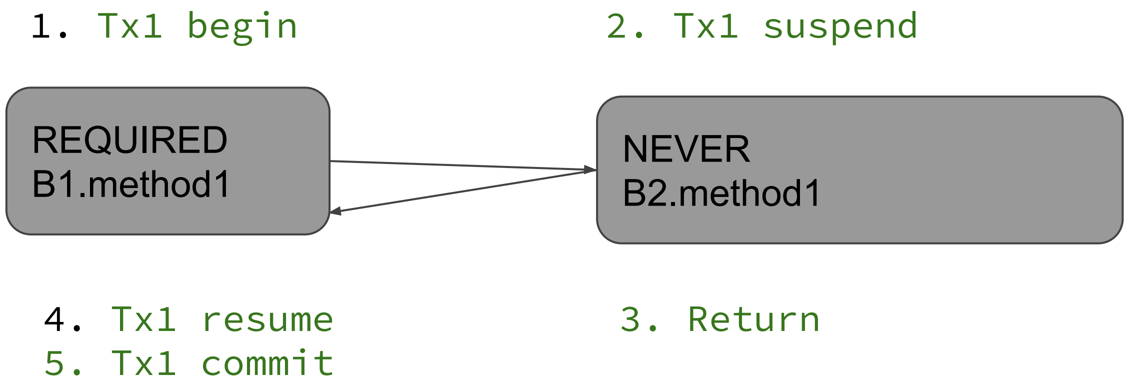


How propagation works: MANDATORY





How propagation works: NEVER



Pitfall: Propagation does not work inside own class

```
@Service public class TransferService {

        @Transactional(value = TxType.REQUIRED, rollbackOn = InsufficientFundsException.class)
        public List<BankAccount> transferMoney(String userFrom, String userTo, BigDecimal amount)
                        throws InsufficientFundsException {

                logService.logTransfer(userFrom,userTo, amount);
                ...
                return Arrays.asList(fromAccount, toAccount);
        }
        @Transactional(value = TxType.REQUIRES_NEW)
        public void logTransfer(String userFrom, String userTo, BigDecimal dec) {...
        }
}
```

Solution create a second Service/Bean

```
@Service public class TransferService {
        @Autowired private TransferLogService logService;

        @Transactional(value = TxType.REQUIRED, rollbackOn = InsufficientFundsException.class)
        public List<BankAccount> transferMoney(String userFrom, String userTo, BigDecimal amount)
                        throws InsufficientFundsException {
                ...
                logService.logTransfer(userFrom,userTo, amount);
                return Arrays.asList(fromAccount, toAccount);
        }
}
@Service
public class TransferLogService {
        @Transactional(value = TxType.REQUIRES_NEW)
        public void logTransfer(String userFrom, String userTo, BigDecimal dec) {...
        }
}
```

Transactions with Spring and TransactionManager

```
public class TransactionManualTest {
  @Autowired
   private PlatformTransactionManager transactionManager;
  ...
  @Test
   void transferManualTest() throws Exception{
      DefaultTransactionDefinition definition = new DefaultTransactionDefinition();
      definition.setIsolationLevel(TransactionDefinition.ISOLATION_REPEATABLE_READ);
      definition.setTimeout(3);

      TransactionStatus status = transactionManager.getTransaction(definition);
      try {
          accService.withdraw("david", new BigDecimal("1000"));
          accService.deposit("andre", new BigDecimal("1000"));
          transactionManager.commit(status);
      } catch (Exception ex) {
          transactionManager.rollback(status);
      }
```

Quest

Create a Transfer Service that can withdraw from one account and deposit to another account in one transaction.

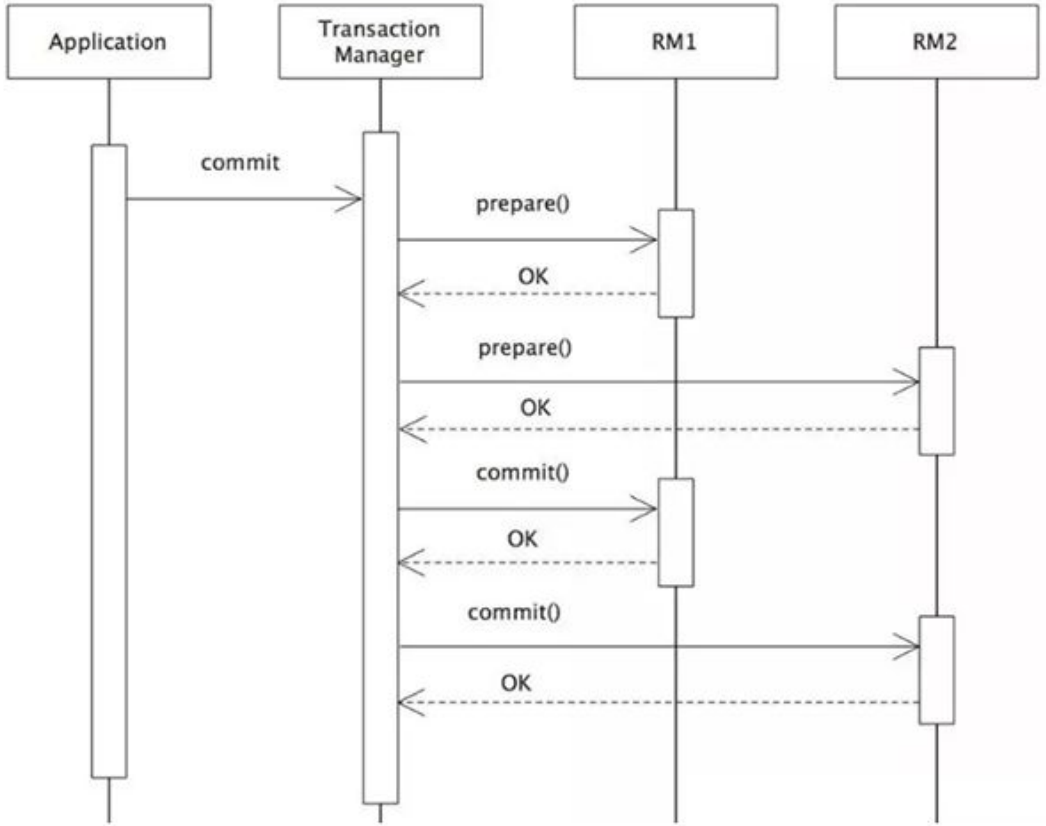
https://github.com/beisdog/wcs-java-transactions-quest.git

wcs-java-tx-spring-boot-quest/

Also create LogService that logs into the database independent of the transaction.

The transfer Service just write a log

Outlook: XA Transactions with 2 Phase Commits



Distributed Transaction (XA Transaction Protocoll)

Needed when you want to mix several systems in one transaction.

E.g:

2 databases or

1 database and 1 message queuing system or something else