entityManager == Persistence Context == first level cache == hibernate session != jdbc connection

https://www.baeldung.com/jpa-hibernate-persistence-context

When talking about a persistence context/entityManager, the first thing coming to mind is lifecycle. Impact factors are scope, open-session-in-view.

Container-managed entity manager:

1. Transaction scoped (transaction finishes, persistence context closed, managed entities detached, if not open-session-in-view, connection back to connection pool)

@PersistenceContext

private EntityManager entityManager;

The lifecycle of a transaction scoped persistence context is within a transaction. Once the transaction is committed or rolled back, the persistence context is closed and all entities become detached.

In a controller:

@GetMapping("/one")

**private** Department getDepartment() {

Thread t = **new** Thread(() -> {

Department d = departmentService.findById(1);

System.***out***.println("In created thread " + Thread.*currentThread*().toString() + ": " + entityManager.contains(d)); -> false

});

t.start();

**try** {

t.join();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

**return** **null**;

}

Testing:

@GetMapping("/one")

**private** Department getDepartment() {

Department d = departmentService.findById(1);

System.***out***.println("In TestController: " + entityManager.contains(d)); -> true (The persistence context is opened once it’s needed in a transaction and kept open for the entire lifetime of this request thread since open-session-in-view is true)

**return** **null**;

}

But Jpa has a configuration:

<https://stackoverflow.com/questions/1103363/why-is-hibernate-open-session-in-view-considered-a-bad-practice>

spring.jpa.open-in-view=false (it defaults to be true)

/\*\*

\* Register OpenEntityManagerInViewInterceptor. Binds a JPA EntityManager to the

\* thread for the entire processing of the request.

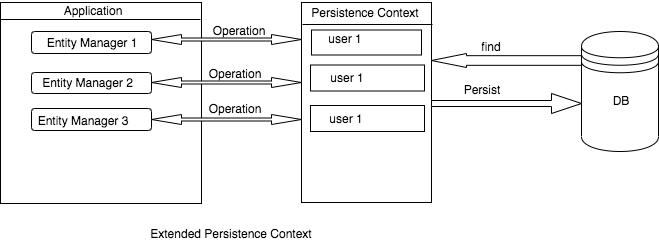
\*/

(2) Extended scoped (transaction finishes, persistence context not closed, managed entities still managed, connection back to connection pool)

@PersistenceContext(type = PersistenceContextType.EXTENDED)

private EntityManager entityManager;

Each container managed entity manager is associated with a different persistence context. It can attach an object into persistence context without a transaction. When used in a transaction, all cached operations (not only from operations in this transaction) in this persistence context will be flushed in that transaction.



Testing:

Component A {

@PersistenceContext(type = PersistenceContextType.***EXTENDED***)

**private** EntityManager entityManager;

**public** VersionTest getVersionTestOneNative() {

Optional<VersionTest> ret = testVersionService.getVersionTestOneNative();

VersionTest vt = ret.get();

System.***out***.println(entityManager.contains(vt)); -> false, vt is returned from a transaction scoped entityManager

// entityManager.find(VersionTest.getClass()); -> return the same object each time though called from different threads. Different from that in Component B.

**return** ret.orElse(**null**);

}

}

Component B {

@PersistenceContext(type = PersistenceContextType.***EXTENDED***)

**private** EntityManager entityManager;

**public** Optional<VersionTest> getVersionTestOneNative() {

Optional<VersionTest> vtOpt = versionTestDao.findById(1);

VersionTest vt = vtOpt.get();

// this vt object is different when called each time from a different thread.

System.***out***.println(entityManager.contains(vt)); -> false

entityManager.persist(vt);

// entityManager.find(VersionTest.getClass()); -> return the same object each time though called from different threads.

System.***out***.println(entityManager.contains(vt)); -> true

**return** vtOpt;

}

}

https://stackoverflow.com/questions/2547817/what-is-the-difference-between-transaction-scoped-persistence-context-and-extend

## 5.6 Container-managed Persistence Contexts

(...)

A container-managed persistence context may be defined to have either a lifetime that is scoped to a single transaction or an extended lifetime that spans multiple transactions, depending on the PersistenceContextType that is specified when its EntityManager is created. This specification refers to such persistence contexts as transaction-scoped persistence contexts and extended persistence contexts respectively.

(...)

### 5.6.1 Container-managed Transaction-scoped Persistence Context

The application may obtain a container-managed entity manager with transaction-scoped persistence context bound to the JTA transaction by injection or direct lookup in the JNDI namespace. The persistence context type for the entity manager is defaulted or defined as PersistenceContextType.TRANSACTION.

A new persistence context begins when the container-managed entity manager is invoked[36] in the scope of an active JTA transaction, and there is no current persistence context already associated with the JTA transaction. The persistence context is created and then associated with the JTA transaction.

The persistence context ends when the associated JTA transaction commits or rolls back, and all entities that were managed by the EntityManager become detached.

\*\* IMPORTMANT \*\*

\*\* If the entity manager is invoked outside the scope of a transaction, any entities loaded from the database will immediately become detached at the end of the method call. If the invoked method is read/update, then TransactionRequiredException will be thrown.

### 5.6.2 Container-managed Extended Persistence Context

A container-managed extended persistence context can only be initiated within the scope of a stateful session bean. It exists from the point at which the stateful session bean that declares a dependency on an entity manager of type PersistenceContextType.EXTENDED is created, and is said to be bound to the stateful session bean. The dependency on the extended persistence context is declared by means of the PersistenceContext annotation or persistence-context-ref deployment descriptor element.

The persistence context is closed by the container when the @Remove method of the stateful session bean completes (or the stateful session bean instance is otherwise destroyed).

(...)

This is thread based:

@Transactional

**public** List<Employee> getAllEmployee() {

Employee employee = entityManager.find(Employee.**class**, 1);

entityManager.contains(employee);

entityManager.merge(employee);

// entityManager.merge(employee);

Thread t = **new** Thread(() -> {

tempService.show(employee);

});

t.start();

**try** {

t.join();

} **catch** (Exception e) {

// **TODO**: handle exception

}

**return** **null**;

}

@Service

**public** **class** TempService {

@PersistenceContext

**private** EntityManager entityManager;

@Transactional

**public** **void** show(Employee employee) {

Employee employee1 = entityManager.find(Employee.**class**, 1);

System.***out***.println(entityManager.contains(employee1));

-> true

System.***out***.println(entityManager.contains(employee));

-> false

}

///////////////////////////

**public** **void** show(Employee employee) {

Employee employee1 = entityManager.find(Employee.**class**, 1);

System.***out***.println(entityManager.contains(employee1));

-> false

System.***out***.println(entityManager.contains(employee));

-> false

}

}