**Entitások életciklusa**

**példányosítás-> new állapot, persistelés->managed állapot = JPA hatáskörében van; egyirányú kapcsolat az adatbázis és a példány között; managed állapotban módosítás -> leképződik az adatbázisban is; betöltésnél (query vagy find) alapból managed lesz; detached() vagy close() vagy clear() -> detached állapot, nincs további kapcsolat az adatbázisal, ezt vissza lehet tenni managedbe merge()el -> ilyenkor új objektum lesz az eredeti értékeivel; remove()-> csak flag, tényleges törlés csak commitkor, ha van előtte egy persist, újramanaged)**

*A JPA hogyan manageli az entitások állapotait, milyen állapotban jöhetnek létre, és milyen más állapotba mehetnek át? Persistence context fogalma*

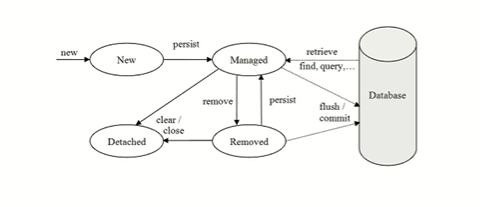
Amikor **példányosítunk** egy entitást Java kódból new-al, akkor egy **new állapotba** jön létre. Ha ezt az entitást aztán **adatbázisba lementjük** az entitymanager.persist()-el, akkor egy ún. **managed állapotba** kerül **-> itt már a JPA hatáskörében van** az adott példány; egy láthatatlan fonál az objektum és a mögötte lévő adatbázis rekord között; **ha a managed állapotban módosítom az entitás bármely atriját, akkor a tranzakció végén a módosítás leírásra kerül az adatbázisba** -> az objektum és az adatbázis rekord között **egyirányú kapcsolat** van

**ha** nem én példányosítok, hanem **betöltök** pl**. find()-al, vagy createQuery()** eredményeként jön létre az adott entitás, akkor **alapból managed** állapotba kerül

ha **ezzel aztán valamiféle műveletet végzek,** pl. meghívom a detached metódust, vagy ha megszűnik/ürítésre kerül a perzisztencia context, akkor **az entitások detached állapotba kerülnek**, **nincs továbbá kapcsolat**/láthatatlan fonál az adatbázis és az objektum között, **módosítás már nem képződik le az adatbázis rekordon**

ha azt szeretném, hogy ez **újra managed állapotba kerüljön**, akkor meg kell hívni a **merge() metódust**; ilyenkor nem a régi objektum kerül visszacsatolásra, hanem egy **új objektumot fog létrehozni; ez már managed** állapotban lesz, ugyanazokkal az értékekkel, mint amivel az eredeti objektum is rendelkezett

ha meghívom a **remove()-t**, akkor **nem azonnal kerül törlésre** az adatbázisban, **csak egy flaget** kap;(=meg lesz jelölve törlésre) **amikor vége van a tranzakciónak, akkor fogja** a JPA implementáció **törölni**; de ha **időközben** meggondoljuk magunkat és meghívjuk a **persist()**-et, **újra visszatér managed állapotba**, és nem fog törlésre kerülni



**Persistence context fogalma**

**(futásidejű, megnyíl-lezáródik, managed állapotú entitások összegsége a memben; háttérben kapcsolat az adatbázissal; hossza tranzakciónyi vagy azon átível, előbbi használata; detached()el vagy clear()el kerül ki innen az entitás, vagy mert a tranzakció véget ér; visszamergelés commit előtt-> ha kliens oldalra is átküldjük és utána mergel managedre tesszük és új persistence contextet nyitunk és és merge() -> inkább töltsük be az entitást és és setter, mert kontrolálni tudjuk vele a változásokat, a mergenél minden atri értéke feülíródik**

**futásidőben** létező fogalom(dinamikus), akkor kerül megnyitásra és lezárásra is

**a managed állapotú entitásoknak az összessége**; bent vannak a **memóriában**, a JPA tud róla és a háttérben **össze vannak kötve** az adatbázissal

meddig tart egy persistence context? a **hossza vagy tranzakciónyi** (kinyílik a **tranzakció indulásakor és commit hatására szűnik meg**); **vagy** lehet **tranzakciókon átívelő** is; **a legtöbb esetben a tranzakciónyisat** használjuk -> megnyílik a tranzakció, létrejön egy memória terület, ahova **persist-el vagy find-al vagy queryvel kerülnek be az entitások**, és egészen odáig itt vannak, **amíg manuálisan ráhívjuk a detached-et**, amivel kivesszük a persistene contexből, **vagy clear** metódussal kiürítjük a teljes contextet; másik esetben magától **megszűnik a persistence context** mert **véget ér a tranzakció**

Amikor **él a persistence context és még nincs commit,** és a **detached-el leválasztottunk** egy entitást, azt **vissza is tudjuk mergelni;** ilyenkor **új entitás jön létre managed állapotban**, a régi objektum értékeivel**; mikor használjuk ezt?** többrétegű alkalmazásnál, ha betöltjük az entitást, majd lezárul a persistence context, és a **detached állapotban lévő entitást átküldjük kliens oldalra, ott dolgozunk vele, módosítjuk a tartalmát, de mivel detached állapotban van, nem fog visszaíródni az adatbázisba, hanem sima java objektumként viselkedik; hogy a módosításai visszakerüljenek adatbázisba, érdemes nyitni egy új persistence contextet, majd ezt a detached állapotban lévő entitást visszamergelni**, ezáltal a meglévő módosításokkal frissülni fog az adatbázis tartalma; erről megoszlanak a vélemények, hogy ez jó-e; **sok problémát úszunk meg, ha nem használjuk a merge-t;** **inkább betöltjük az entitást a finddal, majd setterekkel beállítjuk CSAK azokat új értékeket, ahol módosítást akarunk az adatbázisban; a mergenél** ez nem garantált, **ilyenkor a detached entitás összes atri értéke felülírja a betöltött entitás atrijainak értékeit**

**Lifecycle események**

**(lifecycle – állapotváltozáskor fusson le egy metódus; annotácókkal érjük el; ritka, pl. ha mezőérték számítása bonyolultabb, cachelési célok; flush – commitig is történjen adatbázisba írás, ne csak egyben a végén, de ezt is lehet roollbackelni; persistence context és persistence unit definíciók))**

**állapotátmenetet esetén** különböző **metódusokat**, kódrészleteket tudunk **meghívni** **annotációk alapján**

az entitásban metódusokat deklarálunk és annotációval látjuk el

@PrePersist, @PostPersist – az ilyen annotációval ellátott metódusok a persistálás előtt vagy után futnak le

@PreRemove, @PostRemove – törlés előtt vagy után

@PreUpdate, @PostUpdate – módosítás előtt vagy után

@PostLoad – entitás betöltése után

ritkán használjuk; nem érdemes bennük üzleti logikát implementálni; **ha egyszer ki kell számolni a mező/atri értékét, valami bonyolultabb módon, és cacheelni szeretnénk,** el szeretnénk tárolni az objektumunkba**; tipikusan tranziens mezőben** megoldva; **ilyenkor érdemes az ilyen számolást egy lifecycle eseménybe implementálni**

**Flush metódus**

**amíg megnyílik egy tranzakciónyi persistence context, a módosítások a commitig nem történik adatbázisba írás;** **ha közben is szeretnénk azt, hogy kiírásra kerüljenek, akkor** használjuk a flush()-t; ilyenkor közbe is leírásra kerülnek a módosítások, de csak az adott tranzakción belül; **de egy rollback ezeket vissza fogja szedni**

**entityManager.flush()-al**

**A peristence context egy futás idejű dolog; addig él, amíg az alkalmazás, benne a managelt entitásokkal.**

**A persistence unit egy statikus dolog, a persistence.xml állomány, ami a JPA-t konfigurálja.**

**Entitások életciklusa – gyakorlat**

*Hogyan működik a persistence context, Milyen összefüggés van az entity manager és a persistence context között, az entitások állapotátmenete*

**a persistence context egy EntityManager példányhoz kapcsolódik; ha példányosítok egy managert létrejön egy pc; ez tranzakció határig tart vagy azokon akár át is ívelhet; egy pc egy entityManager!**

1. *új test metódus: testSaveEmployeeChangeState()*

new Employee(); employeeDao.saveEmployee(); setName(xx); modifiedEmployee = .findEmployeeById(); sout name; futtatás

miket látok? **hibernate log szerint egy insert történt, nem történt select;** az adatbázisban nem módosult a név, miért? Az EmployeeDao létrehozásakor létrehozok egy entitiymanagert, amihez tartozik egy persistence context; ez a teszteset futásáig él, tartalmazza a managelt entitásokat; **ha létrehozok egy új példányt, az new státuszba kerül, nem tud róla a JPA**, még nincs managelve; a save-el managelésre kerül, ott lesz tranzakció és lesz commit; de utána a settel nincs tranzakció

**select azért nem fut le, mert az adott pc még él, és abból szedi ki**, a memóriából: ezt igazolja a sout employee == modifiedEmployee is

**ez így nem jó működés; kerüljön az adatbázisból visszaolvasásra és a név kerüljön módosításra:** **olyan entitymanager kell majd mindig, ami új persistence contexttel rendelkezik**; Spring ezt tudja; java SE-ben pedig entityManagerFactory injection kell a teszt osztályba és mindig új entitymanagert kell belőle gyártani a tesztek elején, majd close();

új futtatásra módosított az adatbázison is és nem a memóriából dolgozott a findEmployeeById()kor select-el dolgozott a Hibernate; a két példány referenciája sem egyezett már meg

1. *Detach működése*

első körben az objektum detached állapotban volt, azért nem került be a módosítás az adatbázisba; hogyan lehet ez merge()el visszacsatolni?

*új metódus void updateEmployee(employee)* ->Dao-ba; új entitymanager entitymanagerfactoryból; majd entitymanager.merge(), begin() és commit() közé -> a paraméterként átadott detached entitást visszacsatolja

*testMerge()* -> new Employee(), save(), setName(), dao.updateEmployee(employee)

futtatás; név módosult az adatbázisban is

**nem a paraméterként átadott entitást csatolja vissza, hanem visszaad egy új entitást, ami már managed;** ha ezen hívok egy új setName()-t, az már szerepelni fog az adatbázisban (\*\*os név)

**lekérdezés hatására lezárásra kerül a pc**, majd a felületen updatelek, azt visszaküldöm, **mergelem, akkor az összes módosítás felülíródik** -> ezért **veszélyes a merge; jobb az explicit megadás**, hogy mit akarok módosítani; majd betöltöm az entitást, és magam állítom be a módosításokat

1. *Életciklushoz tartozó események működése*

*új metódus Employee-ba: void debugPersist() ->*sout name + id

ha lefut a persist, automatikusan hívódjon ez meg: *@PostPersist*

1. *Flush() működése*

Daoba ú*j metódus: void updateEmployeeNames()*

createquery mindenre és listába mentés; ezek már managedek ilyenkor; majd .begin és close; ciklus setname \*\*\* + sout „Módosítva”

*void testFlush(*); ciklusban sok employee lementése, majd updateEmployeeNames() -> először a sok „Módosítva” üzi de csak utána jönnek az updatek; **csak commitra generálódik le az update a végén egy csomagban**

módosítás: c*iklus végére egy .flush()* -> egyenkét fut le az update; egy select közben már a módosítottat venné ki; persze az egészet még mindig lehetne rollbackelni

**Kérdések**

*Milyen állapotati vannak az entitásoknak?*

*Mit jelent a managed állapot?*

*Managed állapotban mi történik a módosított objektummal?*

*Betöltéskor milyen állapotba kerül az objektum?*

*Mit jelent a detached állapot és mikor jön létre?*

*Hogy kapcsolok vissza egy detached entitást (= legyen újra managed)? Mi történik ilyenkor? Mikor használjuk ezt? Mi a jobb gyakorlat ennél?*

*Mi történik az entitással remove() meghívásakor?*

*Mi az a persisetence contect? Mi a különbség a PU-hoz képest?*

*Meddig tart egy persistence context? Hogy kerülhet be egy entitás a contextbe és hogy kerülhet ki?*

*Mik azok a lifecycle eseményke és milyen fajtái vannak? Milyne számításokra érdemes használni?*

*Mi az a flus() metódus?*