**Lekérdezések**

**JPQL**

**(JPQL=oo lekérdező nyelv, entitásokon és atrikon dolgozik nem táblákon és oszlopokon, platformfüggetlenség: háttérben a jpql átfordítása adatbázisspecifikus sqlre, ezt ki is lehet naplózni;; perzisztens mezők: oszlopként az adatbázisba, kapcsolati mezők; identification variable: lekérdezéskor entitáshoz deklarált változónév; path expression: bejárási út az atrikon keresztül, nem kell join, objektum gráf)**

A lekérdezések nyelve JPAban nem SQL, hanem egy, a JPA szabványban definiált nyelv, a JPQL

**a JPQL objektumorientált lekérdező nyelv;** nem a táblákon, hanem **az entitásokon és azok attribútumain dolgozik**

a JPQL nyelv biztosítja a **platformfüggetlenséget**, mert a háttérben a JPA implementáció **átfordítja a JPQL utasításokat az adott adatbázis specifikus SQLre**

**A JPQL tulajdonságai**

az entitások azon mezői, amik konkrétan oszlopként az adatbázisba kerülnek, azokat **perzisztens mezőknek**, amelyek pedig valamiféle kapcsolatot reprezentálnak, azokat **kapcsolati mezőknek** hívjuk (@OneToOne, @OneToMany és @ManyToMany annotációk rajtuk)

**identification variable**: lekérdezés írásakor az **entitás** **deklarálása** mellett egy **változó** **nevet** is deklarálunk

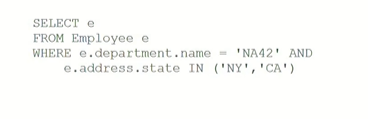
**a JPQL utiból a JPA implementáció egy SQL lekérdezést generál,** ezt kis is lehet **naplózni**, debug céllal ki lehet próbálni egy sql kliensbe megfelelő paraméterezéssel

**JPQL példa Path expression példán**

az identification variable-el mintha deklarálnánk egy Employee típusú e változót

WHERE után: objektumorientáltság előnyei már jelentkeznek: atribútumra kérdezünk rá

**path** **expression; egy bejárási utat tudunk megadni az atrikon keresztül;** így járjuk be az **objektum gráfot, nem kell explicit módon a join-t kiírni**

****

**Egyszerű lekérdezés, streammel, egy találat, paraméterezett lekérdezés**

**(egyszerű: entityManager.createquery(String JPQL úti, entitás osztálya).getResultList(); streames: getResultList() helyett getResultStream(); biztosan tudjuk hogy egy találat(ha nem, exception lesz): getSingleResult(); paraméterezett: paraméter előtt kettőspont, setParameter(paraméter név, érték), kifejezés eltárolása**

**Egyszerű lekérdezés**

hogy használhatunk kódból egy egyszerű lekérdezést?

az **entityManager createQuery**() metódusával, **paraméterként** átadva neki **Stringként a JPQL utasítást és az entitás osztályát**, hogy ne kelljen castolni



**Egyszerű lekérdezés stream eredménnyel**

.getResultList() helyett .**getResultStream()**, ami a listát streamként adja vissza; ezzel **csak azokat az eredményeket hozzuk át az adatbázisból a memóriába, amikre nekünk szükségünk van**

****

**Egy találat**

amikor tudjuk, hogy **egy lekérdezésnek biztosan csak egy találata** lesz

**.getSingleResult()** meghívásával; **ha mégse** egy eredmény vagy nincs eredmény, **kivétel** lesz



**Paraméterezett lekérdezés**

a paramétereket kettősponttal kezdjük; majd ezt **cseréljük ki egy konkrét értékre a .setParamter() hívással;** ennek első **paramétere** a **paraméter neve, második az értéke**; ennek hatására lesz a kicserélés

a **paraméterezett queryt eltárolhatja a jpa implementáció és cachelésre** is jó



**Criteria API, MetaModel API, Lapozás, Definiálás named query-ként, Join, Objektum tömb, Projection query, Query timeout**

**(Criteria API: lekérdezés programozott előállítása, hogy ne gépeljük el a Stringes JPQL-t: CriteriaBuilder, CriteriaQuery, Root; MetaModel API: még a Stringes nevet is konstansra cseréljük;Lapozás: createQuery() után . .setFirstResult() és setMaxResults(); named query: Stringes query annotációba mentése, helyesség ellenőrzése már indításkor, meghívás createNamedQuery()vel, futási idejű létrehozása: addNamedQuery()vel, TypeQueryként létrehozva; Join path expresisonnel és identificaiton variableökkel; getResultList() egy objektumtöböt ad vissza atrik felsorolása esetén a selectben; projection query: DTOt ad vissza a query, ilyenkor teljes minősített névvel kell megadni az osztály nevét; timeout: hintként átadva, dobjon exceptiont ha exceedeli)**

**Criteria API**

A **lekérdezéseket** eddig **Stringbe** írtuk; **elgépelési lehetőség,** ami csak **futási időben derül ki**

lehetőség van az adott **lekérdezések programozott előállítására** (nem egyenlő String konkatenációval!)

**Criteria API: lekérdezés összeállítása java metódusokkal**

ez típusbiztos, mert ha a metódusneveket **elgépeljük**, **fordítási idejű hiba** lesz

****

public Employee findEmployeeByName(String name) {  
 EntityManager entityManager = entityManagerFactory.createEntityManager();  
 CriteriaBuilder cb = entityManager.getCriteriaBuilder();  
 CriteriaQuery<Employee> c = cb.createQuery(Employee.class);  
 Root<Employee> emp = c.from(Employee.class);  
 c.select(emp).where(cb.equal(emp.get("name"), "John Doe"));  
 Employee employee = entityManager.createQuery(c).getSingleResult();  
 entityManager.close();  
 return employee;  
}

**MetaModel API**

de a **name atrit még itt is el lehetne gépelni**, mert String**; ennek kiküszöbölésére** jött létre ez az API

el tudjuk kerülni a beégetett karakterláncokat

későbbi **refaktorálásra is biztos** a működés

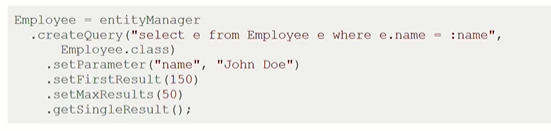
az **entitások alapján** kigenerálhatunk egy **canonical metamodellt;** kigenerált osztályok különböző mezőkkel**, konstans értékek**, amiket felhasználhatunk lekérdezésekben; a „name”t kicserélhetjük a konstansra

**Lapozás**

JPA egy másik tulajdonsága

.**setFirstResult()** – hányadik indexű entitástól adja vissza az entitásokat

és **setMaxResults()** hány darab entitást adjon vissza

****

**Definiálás named query-ként**

=**nevesített lekérdezések**

**Stringként adjuk meg a lekérdezéseket,** de nem metóduson belül, hanem **annotációként**, névvel

**meghívás** nem a sima creatrQuery()vel hanem **a createNamedQuery()vel, paraméterként** a **query nevét** átadva

az **annotáció az entitásra kerül**

**gyorsabb** futást eredményez; elcasheli indításkor

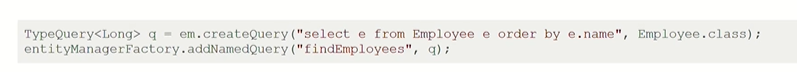
**szintaktikai helyesség ellenőrzése már az alkalmazás indításakor** végbemegy

**hátrány**, hogy a lekérdezés tartalmát, **hogy lássuk, el kell ugranunk az entitásra,** hogy megtaláljuk; fejlesztő eszközök azért ebben segítenek (felugró ablak)



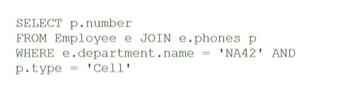
**Hogyan lehet egy named queryt futási időben létrehozni?**

Persistence contextbe hozzáadás **addNamedQuery()vel**; lekérdezés **eltárolása TypeQueryként**



**Join használata JPQLben**

**path expressionnel** (e.phones**), identifiction variable**-ökkel (e és p)

****

**Objektum tömb**

**Amikor a select után attribútumokat sorolunk fel**, a **visszatérési érték** egy entitás esetén egy **objektum tömb** lesz, a getResultList() pedig egy objektum tömböket tartalmazó listát fog visszaadni

****

**Projection query**

**nem entitást ad vissza, hanem DTO-t;**

**teljes minősített névvel kell megadni az osztály nevét**, konstruktorparaméterként az atri érétkeket; a háttérben lefut a select, visszaadja az atri érétkeket majd **reflectionnel példányosít** egy EmployeeData-t==**DTO**?)



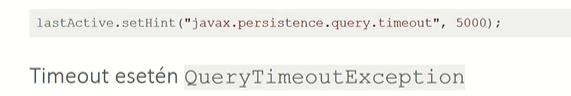
**Query timeout**

A **JPA implementációknak** lehet különböző **hinteket adni**, amikkel **befolyásolni** tudjuk az adott **lekérdezés működését**

ilyen hit lehet pl. a **timeout**; **paraméterként minisecben** adjuk át az időt

ha a **lekérdezés ezen időn belül nem megy végbe,** **exceptiont** fogunk kapni

de a timeoutot az adatbázisnak és a jdbc drivenrek is támogatnia kell



**Lekérdezések – gyakorlat**

1. *Criteria API használata*

EmployeeDaoban új metódus findEmployeeByName(String name); em és em.getCriteriaBuilder(), majd cb.createQuery(class), c.from(class), c.select(root).where(cb.equal(root.get(„name”),name)); createQuery(c).getSingle..;

új test void testFindEmployeeByName(); saveemp, betöltés findempbyname új metódusommal, majd assert; milyen sqlt generált le?

1. *Lapozás implementálása*

új metódus daoba: List<E> listEmployees(int start, int maxresult); új em, jade m.createQuery(…)..setFirstResult().setMaxResults()…close(); return employees

void testPaging(); ciklusban generálok 100 új employeet név +i-vel, saveemployee, majd assert listEmployees(50, 20)

1. *Named query használat*

@NamedQuery(value =„listEmployeees, query = (jpql string); createQuery helyett

createNamedquery(„listemployees”)

1. *ParkignPlace számok lekérdezése egy alkalmazottra*

új DAO metódus: int findParkingPlaceNumberByEmployeeName(String name); új em, em.createQuery(„**select p.number from Employee e join e.parkingPlace p where e.name=:name”**).getSingle, close

test testFindNumber(), new emp, new parkingplace, emp save, setprkingplace, előtte kell parkingdao az osztályba, betöltés, assert rá

1. ***Több atrira rákérdezés***

**daoba új List<Object[]>** listEmployeeBaseData; em.createQuery(sleect e.id, e.name…)…

void testBaseData(), new emp, save, visszaolvasás új metódussal, assert

1. *Projection query, kapjuk vissza az objektumot dto-ként*

EmpBaseDto osztály id és name, getset, konstruktor

új met daoban: listEmployeeDto(); new em, em.createQuery(+select jpa.EmpBaseDataDto()e.id, e.name) from Employee e”).getResutsList

void testDto(); 2 employee save, majd betöltés új metódussal, assert sizera és 2. nevére, kell a seletcbe order by

**Kérdések**

*Mi az a JPQL?*

*Min dolgozik?*

*Hogy biztosítja a platforfüggetlenséget?*

*Mik azok a perzisztens és kapcsolati mezők?*

*Mi az az identificaiton variable?*

*Mi az a path expression?*

*Hogy indítunk kódból egy egyszerű lekérdezést? Milyne paraétereket kell megadni?*

*Milyen metódusokat használunk a láncolt hívásban egy találat és több találat esetén?*

*Hogyan érjük el, hogy a lekérdezés visszatérési értékéből pl. csak az Employeek nevét kapjuk vissza listában?*

*Hogy paraméterezzük a lekérdezést?*

*Hogyan tudjuk a lekérdezéseket programozottan előállítani? Milyen APIk vannak erre?*

*Hogyan tudunk lapozni a lekérdezés eredményén belül?*

*Mik azok a nevesített lekérdezések, és hogyn adefiniáljuk őket? Mi az előnye és hátránya a használatának? Hogyna lehet futási időben létrehozni?*

*Hogy használjuk JOINT JPQL-ben?*

*Mi lesz a visszatérési érték, ha a SELECT után attribútumokat sorolunk fel?*

*Mi az a projection query, és hogy adjuk meg?*

*Hogy adjuk meg, hogy a lekérdezés ha nem megy bizonyos időn belül végbe, kapjunk exceptiont?*