**Spring Data JPA**

**ORM eszköz = automatikus megfeleltetés object-adatbázis rekord; JPA API és implementációk, SpringBoot -> Spring Data JPA, még egyszerűbb; leggyakrabban használt CRUD és egyéb műveletek tartalmazó repok automatikus generálása; interface kiterjesztésével, generikussal(entitás és azonosító); query by example tulajdonság: lekérdezés átadott objektum alapján**

**Spring Data JPA**

A JdbcTemplate használata után a következő lépés, ha az adatbáziskezeléshez **ORM eszközt** használunk

ORM = object relational mapping; keretrendszer, ami **automatikusan megfelelteti a java objektumokat az adatbázis rekordjainak**

**leggyakoribb ORM eszköz Javaban a JPA**

**a JPA egy API**, aminek különböző implementációi vannak

legelterjedtebb implementációja a **Hibernate** és az EclipseLink

A JPA-t nem feltétlen érdemes direktbe használni **SpringBootos projekt esetén; helyette van egy library, a Spring Data JPA**, ami **a JPA-nak a használatát is még sokkal egyszerűbbé** teszi

**automatikusan** tudunk vele olyan **repository osztályokat generáltatni**, amelyek a **leggyakrabban használt** műveleteket automatikusan tartalmazzák; így nem nekünk kell a **CRUD műveleteket** implementálnunk, **csak meghívnunk kell őket**

olyan **gyakori feladatok is** implementáltak benne, mint pl. a **rendezés**, vagy a **lapozás** kezelése

Segítségével egy **interfacet kell kiterjesztenünk,** **generikus típusként az entitást és az azonosító típusát** megadva; ezzel az interfacehez tartozó **implementációt** **a Spring Data JPA automatikusan elkészíti**

**Query by example tulaidonság: átadunk egy objektumot, aminek ki vannak töltve bizonyos mezői, és azon objektumok lesznek lekérdezve, amelyek mezői hasonlóképp vannak kitöltve**

használatával sok kód implementálástól szabadulunk meg(boilerplate)

**Spring Data JPA használatba vétele**

**pom.xml: spring-boot-starter-data-jpa; JPA annotációkkal entitás létrehozása; saját interface ami kiterjeszti a JpaREpository interfacet; default metódusok és saját implementációk; hogy egy tranzakcióba kerüljenek a repo hívások: service rétegben @Transactional a metódusra; konfigurálás application propertiesben->shows sql**

**pom.xml függőség felvétele: spring-boot-starter-data-jpa;** ez tranzitívan berántja a JPA API-t is és annak default implementációját, a Hibernate-et

innentől kezdve használhatjuk a **JPA annotációt entitások létrehozására**

ha kész az entitás, létre kell hozzuk a **saját interface-ünket, ami a JpaRepository interfacet terjeszti ki**

**az alapba implementált metódusokon túl további metódusok létrehozása annotációkkal** arra vonatkozóan, hogy **hogyan történjen a select**

ha azt akarjuk, hogy **több repository hívás egy tranzakcióba** kerüljön, akkor azt a service rétegben kell megvalósítani, azáltal, hogy a **serviceben lévő publikus metódusra rátesszük a @Trasactional annotációt**

magát a keretrendszert **konfigurálni a application.properties** állományban lehet; sok előre definiált property van; ilyen pl. a **spring.jpa.show-sql=true**, aminek hatására a konzolon látni fogjuk a legenerált sqleket is

**JPAREPOSITORY**

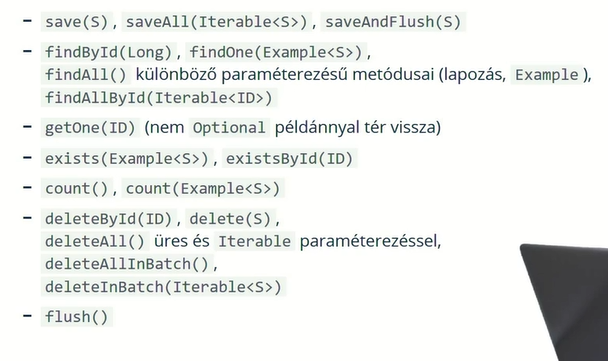
**JPAREP metódusai: findById() Optional; save, saveall, findONe, findAll, exists, count, deleteById, deleteAll, flush**

*Milyen metódusokat tartalmaz?*

sokkal többet, mint az alap JPA-ban az EntityManager

findbyId() Optional-al tér vissza!

az entityManager csak entitás alapján képes törölni, a spring data jpa id alapján is (deleteId())



**Entitás**

*Hogy néz ki ezek alapján egy entitás?*

**kötelező @Entity és @Id**

**@Table(name=..)** entitáshoz rendelt tábla nevének személyre szabása

**@Column(name=..)** – entitás mezőjéhez/attribútumához rendelt oszlop nevének személyre szabása

az alapértelmezetten használt nevek az adatbázisban az osztály és attribútum nevekkel egyeznek meg



**Repository**

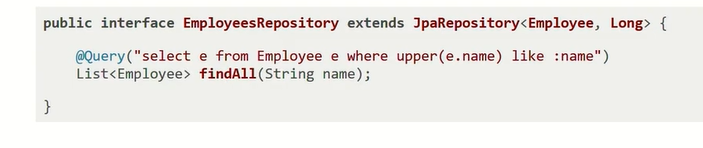
**default metódusokon felüli metódusok: automatikus JPQL legenerálás név alapján, szabályszerűség szerint, vagy explicit megadjuk a JPQL-t, @Query annotációval, metódus paraméterre hivatkozás a queryből**

*Hogy néz ki a Repository?*

**A Repositorynk ki kell terjessze a JpaRepository interfacet, és generikus típusként át kell adni az entitás osztályát, és az id típusát**

ha a **JpaRepositoryban lévő metódusokon felül** szeretnénk még többet, akkor **egyrészt** **név alapján** is tudunk **generálni metódusokat**: **szabályszerűség szerint megadunk egy metódus nevet és a mögötte lévő JPQL kifejezést legenerálja metódus név alapján** a spring data jpa

**másrészt** ha ez nekünk nem megfelelő, **magunk is megadhatjuk a JPQL kifejezést**, ilyenkor a **@Query** annotáció és a **metódus paraméterére magából a queryből tudunk hivatkozni**, a **paraméter neve elé tett kettősponttal**, erre egy példa:

****

**H2, H2 elérés**

**embeddable adatbázis, tesztekre használjuk, mindig drop and create m memben!**

**H2 egy embeddable adatbázis, ami beépül az alkalmazásba és** JVMmel futtatható; ritka használat; ezért a teszteset képes elindítani és leállítani; egy mariadb nem ilyen, annak explicit indítás leállítás kell

mindig újraindul, üres adatbázissal; a mariadbnél drop and create config kell ehhez

*pom.xml függőség:*

****

*applicaiton properties és h2 console config: url!*

jdbc:h2:mem:testdb

**Spring Data JPA – gyakorlat**

1. SpringBootos Commandos employees alkalmazás kiterjesztése
2. pom.xml: spring-boot-starter-data-jpa és h2, devtools?
3. Employeeből legyen entitás; @Entity, @Table, @Id(strategyvel), @Column
4. EmployeesRepository interface létrehozása(+generikus!), ami kiterjeszti a JpaRepositoryt
5. EmployeeDao osztály letörlése
6. EmployeeService-ben a daok átírása employeeRepositoryra
7. metódusok átírása a serviceben:

listEmployees: findall van a spring data jpaban is, maradhat

findemployeebyid():optionalt fog visszadni, ezért orelsethrow

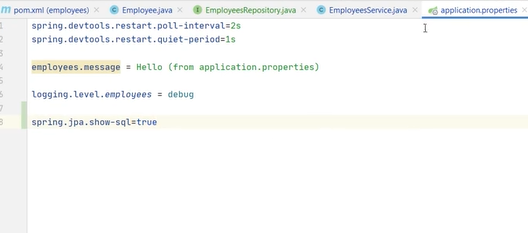
createEmployee(): save-re átírás

updateEmployee: @Transactional, először betöltés id alapján és setter; a **@Transactional a metódust begin() -commit és close() közé teszi**

1. deleteEmployee() és deleteAllERmployees(): : dleeteByid() és deleteAll(); nem kell implementálni, a spring data jpa helyettem implementálja
2. indítás előkészítése

DbInitializer letörlése

application properties-ba a show-sql true-ra állítása



1. indítás, majd http requestek beküldése

GET: a háttérben selectet adott ki; de még nincs a táblában semmi

3 POST beküldése Jane Doeval; insterteket adott ki, és **H2 konzolba** bejelentkezéskor az SQL statement fülön kiadtuk a select \* from employees parancsot és láttuk, hogy benne vannak a táblában

újra GET

2es id-s törlése, 3as nevének módosítása; sql utik a háttérben és h2ben a módosítások csekkolása

**Kérdések**

*Mik az adatbáziskezelés fejlettségi szintjei javaban?*

*Mi a Spring Data JPA?*

*Hogy implementáljuk vele a repositorynkat?*

*A repository milyen műveleteket tartalmaz uatomatikusan és még miket tudunk benne pluszba implementálni?*

*Mi az a query by example tulajdonság?*

*Hogy konfifuráljuk?*

*Hogy érjük el, hogy több repository hívás egy tranzakcióba kerüljön?*

*Hol konfiguráljuk magát a keretrendszert? (pl. ..show-sql=true)*

*Milyne metódusokat tartlmaz a JPARepository?*

*Hogy néz ki az entitás osztály, annotációkkal, lombokkal?*

*Hogy tudunk generálni név alapján a default metóduspokon felül még többet?*

*Hogy tudunk generálni metódusokat saját queryvel?*

*Mi az a H2? Mire használjuk?*