## Egyszerű mentés JPA-val

**(impedance mismatch, JDBC bonyolultsága; ORM eszközök; JPA implementációk; pom dependencyk; peristence xml(pu, transaction type és 4+1 propertyk)**

OOO nyelv -> osztályok; relációs adatbázis -> táblák; **impedance mismatch**: eltérés az oo és az rdbms világ között; koncepcionális és technológiai nehézségek

**JDBC bonyolultsága**: leképzés a két világ között; **resulsetből az értékeket java objektummá és visszafelé**; ez egy automatizálható feladat -> **keretrendszerek** ennek elvégzésére **konfiguráció alapján = megmondom, hogy kell az adott objektumot adatbázisba leképezni és fordítva**

ezek az eszközök az **ORM** eszközök**: object relational mapping; az oo világ és a rdbms világ közötti megfeleltetésért felelős**

Java-ban erre **szabvány** a JPA

JPA több **implementációja**: **Hibernate**, EclipseLink, OpenJPA

**a JPA a JDBCre épül,** háttérben jdbc hívások vannak

használatához **pom.xml dependency** felvétele (**Hibernate**-re) és **a jaxb-nek** is a classpathon kell lennie(java 9ből kivették ezért explicit módon függőségként fel kell venni)

**Konfigurálnunk** kell magát a **JPA**-t is; ehhez ún. **persistence xml-t kell használnunk**; definiálni kell benne egy **persistence unitot(pu),** meg kell mondani, **hogyan történjen a tranzakció kezelés**(”RESOURCE LOCAL” = csak egy adatbázsion belüli tranzakciókat szeretnék használni), és **4 propetrty**: **JDBC driver neve, adatbázis hozzáférés jdbc urlel, felhasználónév és jelszó, hibernate.show\_sql property:** ja a jpa legenerál egy **SQL utasítást**, **azt írja is ki a konzolra**, hogy ellenőrizni is tudjam



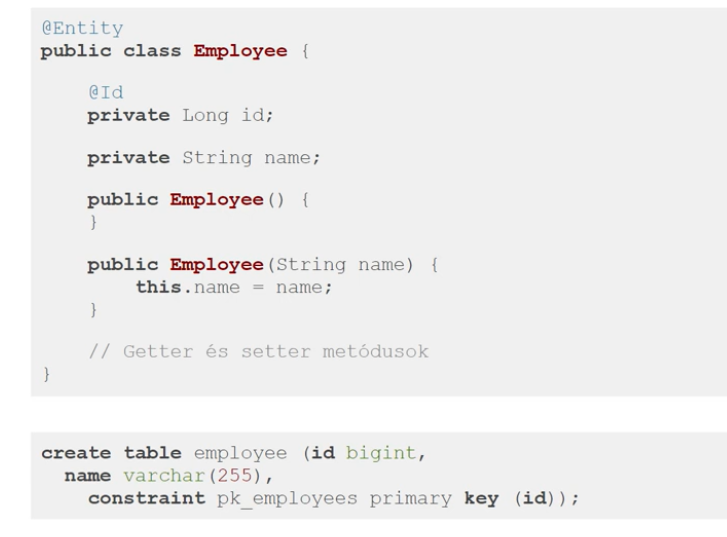
**Entitások**

**(egy az egybe leképzés, @entity és kötelező azonosító atri @Id; adatbázis és táblák legenerálása annotációk alapján, default nevek és primary key, SQL uti megfelelés, személyre szabás @Table-el, @Column-al; azonosítógenerálás @GeneratedValue-al(strategy=GenerationType.IDENTITY)=autoincrement); entitás adatbázisba mentésének lépései: EntytyManagerFactory, EntityManager, .getTransaction.begin(), new Employee, .persist(employee), .getTransaction.commit(), closeok(); összegzés)**

A JPA alapvetően entitásokkal dolgozik; **egy az egybe leképezi őket táblákra**

hogy egy osztály entitás legyen, el kell látni az **@Entity** annotációval és kötelezően rendelkeznie kell egy **azonosító attribútummal**, amin az **@Id** annotáció van

a JPA implementációk az **annotációk alapján** **legenerálják az adatbázisban lévő táblákat**



**alapértelmezett beállításokkal** dolgozik: tábla neve = osztály neve, mezők neve = attribútumok neve + **id** lesz **a primary key**

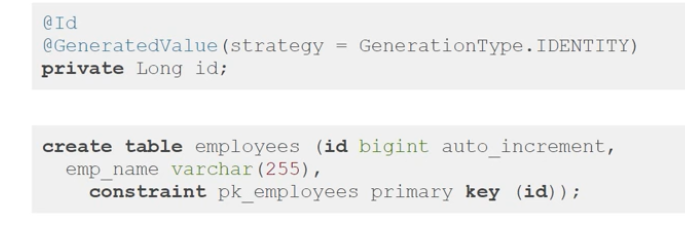
A **neveket személyre** is tudjuk szabni: **@Table, @Column** annotációk



**Azonosítógenerálás**

lehetőség arra, hogy ne nekünk kelljen megadni, **hanem legyen kiosztva az adatbázis által**, az **adatbázis** valamilyen **tulajdonsága** **alapján**

ehhez **@GeneratedValue(strategy=GenerationType.IDENTITY)** annotáció: az adott oszlop típusa legyen olyan, hogy az adatbázis automatikusan töltse fel értékkel, hogyha mi nem adunk neki értéket; felveszi az autoincrement tulajdonságot alapesetben



**Mentés EntityManagerrel**

*az adott osztály egy példányának adatbázisba mentésének folyamata:*

1.EntityManagerFactory létrehozása - át kell adni **paraméterként a persistence unit** nevét, amit az xmlbe írtunk (alapesetben drop and create-el dolgozik!)

2. EntityManager példány létrehozása

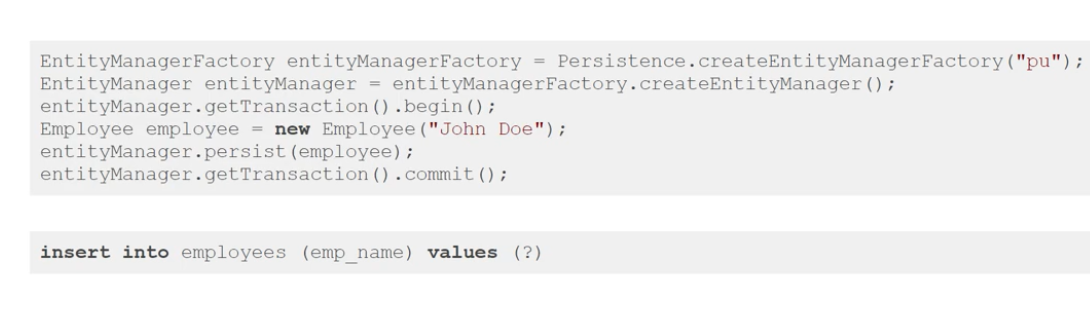
3. Entitymanager.getTransaction.begin() : Hogy az adatbázisba írni tudjunk, meg kell nyitnunk egy tranzakciót

3. lepéldányosítunk egy objektumot

4..persist(employee) - meghívjuk vele a persist metódust(**vigyázni! legyen az entitásnak paraméter nélküli konstruktora is)**

5. .getTransaction.commit() meghívása, **ezzel érnek le az adatbázisba** az adatok

**nem kellett semmi sqlt írnunk, a háttérben az annotációk alapján a jpa legenerálja, és insert utasítást hajt végre; több atri esetén is ugyanennyi lenne a kód és tudná a jpa, milyen sql utival helyettesítse be**



## Egyszerű mentés JPA-val - gyakorlat

* 1. *új üres maven projekt(*név: jpa)
  2. *pom xml*: mariadb, hibernate, jaxb
  3. új entitás felvétele, employees->Employee id/name, noargs, és neves
  4. *Annotációk:*

@Entity és @Id; @Table(name = ”emploees”), @Column(name=”em\_name”)

@Generatedvalue->strategy

* 1. ***resources könyvtárba META\_INF könyvtár és bele persistence.xml,*** bele az előkészített tartalom
  2. *EmployeesMain osztály main()el*; ide a EntityManageres cuccok: factory, manager, begin, persist, commit
  3. *elindítás*; hibernate létrehozza a táblákat
  4. nem állt le; ezért még a main *végére: em.close(), factory.close()*
  5. **az *xmlbe drop and create van*,** vagyis minden alkalommal eldobja a meglévő táblát és újrakreálja és lefuttatja az insertet, majd a végén mostmár le is zár

<persistence version="2.0"  
 xmlns="http://java.sun.com/xml/ns/persistence"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://java.sun.com/xml/ns/persistence  
 http://java.sun.com/xml/ns/persistence/persistence\_2\_0.xsd">  
 <persistence-unit name="pu"  
 transaction-type="RESOURCE\_LOCAL">  
 <properties>  
 <property name="javax.persistence.jdbc.driver" value="org.mariadb.jdbc.Driver"/>  
 <property name="javax.persistence.jdbc.user" value="employees"/>  
 <property name="javax.persistence.jdbc.password" value="employees"/>  
 <property name="javax.persistence.jdbc.url"  
 value="jdbc:mysql://localhost/employees"/>  
 <property name="hibernate.show\_sql" value="true"/>  
 <property name="javax.persistence.schema-generation.database.action" value="drop-and-create" />  
 </properties>  
 </persistence-unit>  
</persistence>

**Kérdések**

*Mi az az impedance mismatch?*

*Mi az az ORM eszköz és mi hívta elő?*

*Mi a JPA, és milyen implementációit ismered?*

*JPA hazálnathoz milyen konfigurációk kellenek? (pom.xml és ersistence.xml)*

*Mi az a java entitás, és hogy hozzuk létre/mi kell hozzá a kódban?*

*Hogyna osztjuk ki az adatbázis által az idt automatikusan?*

*Mi a menete egy objektum lementésének entityManagerrel?*