## Entitások konfigurálása

## (mi kell feltétlenül: @entity és @Id; JPA intelligens alapértékek a nevekre vonatkozóan, személyre szabás;

## sémagenerálási lehetőség az annotációk alapján, ehhez persistence.xmlbe megfelelő property felvétel, generálás scriptbe és generálás adatbázisba,none-create-drop-dropandcreate, vagy magunk írjuk, ilyenkor csak futtatás, flyway alkalmasabb, @Access annotációk FIELD és PROPERTY; Tábla konfigurálás: @Table, name, catalog, schema, constraints; Oszlop konfigurálás: @Column, name, unique, nullable, updateable, columnDefinition, table, length, precision, scale; mezők lementésének támogatása: egyszerű típusok, lob(minden, ami bytetömbe szerializálható) dátum/idő, enum: @Enumerated(EnumTípe.STRING) használat, hogy ne az ordinalt, hanem a name-t mentse le; tranzient mezők(=ne mentse le, csak cashelési célok)

Egy **entitást** **kis számú konfigurációval** kell ellátnunk ahhoz, hogy működőképes legyen -> elegendő csak egy **@Entity** és egy **@Id** annotáció

A **JPA intelligens alapértékekkel rendelkezik:** az entitást az **osztály nevével megegyező** táblába mappeli le, annyi oszlopa lesz, ahány atrija az entitásnak és az **oszlopnevek megegyeznek** az atrik nevével; ez akár **személyre is szabható** (tábla vagy oszlop neve)

**Sémagenerálás**

A JPA lehetőséget biztosít séma generálásra: ***az entitások és konfigurációjuk alapján*** képes legenerálni a táblákat(ill. az annak megfelelő sql kifejezéseket)

nem kell nekünk kézzel megírni a létrehozó utasításokat, és ezek a legenerált kódok egyfajta **prototípusként felhasználhatók; de éles használatuk nem ajánlott**, mert bizonyos dolgok nem szerepelnek benne és pl. korlátozott a séma migrációját illetően -> érdemes inkább erre valami céleszközt használni (Flyway)

a sémagenerálás **forrása** lehet az **annotált entitások** - a konfiguráció alapján fogja kigenerálni az SQL scripteket, de **mi magunk is írhatunk** (ilyenkor nem generálja csak futtatja), de **erre a flyway alkalmasabb**

**célja** lehet script vagy adatbázis: **vagy csak egy fileba kiírja az utasításokat** (ha kiindulásként meg akarjuk nézni, mit generálna ki és nem az adatbázisból indulunk ki, hogy ahhoz hozzunk majd létre entitásokat**) valamint lehet kérni, hogy le is futtassa**, és jöjjenek létre a táblák

**Generálás adatbázisba**

*a konfigurációs paramétereket milyen értékekkel lehet ellátni?*

none(alapértelmezett) – nem történik séma generálás

create: csak generálás, csak tábla létrehozások

drop: séma ürítése

drop-and-create: séma ürítése és generálása is

*a persistence.xml állományba a következő propertyt kell megadni, a választott értékkel:*

**

**Generálás szkriptbe**

## ehhez a …script.action propertyt kelll használni, az elérési útvonalat pedig a create-target propertyvel adhatjuk meg

## 

## Annotációk használata

rá lehet tenni **atrikra(=field access),** **getter metódusokra(property access)**

lehetőleg **csak egyiket használjuk**; ha nekünk ez mégse megfelelő, és mindkettő kell, ezt jelezni kell

-> **mixed access**: **@Access(AccesType.FIELD)** osztály elé és **@Acces(@AccessType.Property)** getter elé

**Tábla konfigurálása**

**@Table annotáció, name attribútummal** (egyéb név legyen ne az osztálynév)

**catalog**, **schema** nevének megadása

**indexek** megadása

**uniqueConstraints** megadása

**Oszlopok konfigurálása**

**@Column** annotáció és **name** attribútum

**unique**

**nullable**

**instertable**, **updateable**, **columnDefinition**(ha az adott oszlopra explicit sql utasítást akarunk rá, hogy az adott mezőben mi szerepeljen; speciális tulajdonság vagy adattípus)

**table, lásd secondary table**; egy **entitás több táblába** is kiírható; bizonyos atrik egyik táblába, bizonyos atirk másik táblába

**length** – szöveges típusnál a VARCHAR hosszát adhatjuk meg

**prescision** és **scale** – a lebegőpontos számok hosszára ad specifikációt

*Milyen típusú mezők lementését támogatja alapból a JPA?*

**Egyszerű típusú mezők**

primitív típusok és burkolóik

String, BigInteger, BigDecimal

**Enumeration = felsorolásos típus**

hogyan kerül lementésre?

**alapértelmezetten az ordinal metódus által visszaadott sorszám**, de ezt felüldeklarálhatjuk az **@Enumerated(EnumTípe.STRING**)-el; ekkor a **name metódus által visszaadott érték;** sorszámot nem érdemes menteni! új érték felvételekor elcsúszik és a visszaolvasás is

**Lob**

= large object; lehet blob(binary) és clob(karakteres); javas megfelelők: byte[] és char[]

**@Lob annotácót** kell rátennünk az adott atrira és fogja a JPA kezelni

bármilyen **szerializálható osztályú** objektum példányt képes elmenteni a JPA, ebben az esetben is @Lob kell; **a kiszerializált bytetömböt fogja elmenteni az adott mezőbe**

**Dátum és időkezelés**

ezredmásodperc pontosságú tárolás a java-s formátumoknál; ha ezeknek csak az idő és dátum megfelelője kell, a **@Temporal** annotációval megadhatjuk, hogy **mely részét szeretnénk tárolni**

a **util.Date** nevével ellentétben időt is tárol, ezért létrehozták az **SQL megfelelő típusokat**: **java.sql.date, java.sql time** és **java.sql timestamp**

**JPA 2.2től** tudja/támogatja a **LocalDate, LocalTime, LocalDateTime, OffsetTime OffsetDateTime** típusokat is

**Tranziens mezők**

**Bizonyos atrikat nem akarunk lementeni**, mert pl. **csak cashelési célokat** szolgált; pl valami értéket csak ki akarunk számolni de mást vele nem

**transient** kulcsszó vagy **@Transient** annotációval

## Entitások konfigurálása – Gyakorlat

## *sémagenerálás és entitások konfigurálása*

## *tesztesetbe kikommenteljük a flyway inicializálást*, tegye meg ezt a jpa, az entitásokon lévő annotációk alapján -> *persistence.xml; új property:* javax.persistence.schema-generation.database.action; value:drop-and-create; indítás: csekkolta milyen sql utikat adott ki, jók-e a nevei a táblának, oszlopoknak., id generált-e, elsődleges kulcs-e

## *konfigok - @Access*

## atrikra tettük az annotációkat; ha getterekre settekre rakjuk az annotációkat, akkor reflectionkor azok lesznek meghívva; ha van bennük logika, bekavarhat; ne legyen ezekbe logika konvenció szerint!

## ha mixelnénk a kettőt(atrira é geterre is), ez se javasolt; ha mégis: ha a getterre rakunk egy @Column-t, meg kell neki mondani a getternél, hogy az Acces type property legyen, az osztály előtt pedig kell majd egy Acces.type field-es annotáció

## *konfigok - @Table és @Columnnál* milyen egyéb atrikat lehet átadni(=?); pl length=200, , nullable=false); újrafuttatás és csekk, hogy milyen sql utit generált?

## Azonosító: nagy Long és kis long id is ok, mint atri; Long azért jobb, mert ha van értéke, akkor az adott példány már perzisztált, ha null, akkor még nincs kitöltve.

## *Enumeration:* private enum EmployeeType (FULL\_TIME, HALF\_TIME) – az osztály elejére!! majd private EmployeeType atri; frissített, bekeült a type; elsőre 0(=ordinal); átírtuk: @Enumerated(EnumType.String)

## *LocalDate atri* dateOfBirth + getter, setter + konstruktor, majd teszt esetből meghívás: testEmployeeWithAttributes(); ebben saveEmployee(név és type és date); majd listEmployee() és get(0), assert

**Kérdések**

*Milyen minimális konfigugrációvak kell elllátni az entitásokat?*

*Milyen alapértékekkel konfigruál a JPA? Hogy lehet ezt személyre szabni?*

*Milyne módjai vanak a sémagenerálásnak? Mi lehet a forrása? Mi lehet a célja?*

*JPA áltlai sémageneráláshoz milyen configok kellenek? Mi akkor akülönbség, ha csak szkripbe akarjuk kimenteni a generált sémát?*

*Milyen annoitációk vnnak a táblák konfigurálására?*

*Milyen típusú mezők lementését támogatja alapból a JPA?*

*Mik azok a tranziens mezők és hogy jelöljük?*