**Elsődleges kulcs**

**(kötelező az egyedi azonosító, min requirement= @id egy atitra = elsődleges kulcs; ált egész szám; automata generáláshoz @GeneratedValue és stratégia : AUTO, TABLE, SEQUENCE, IDENTITY=autoincrementtel!;személyre szabások @\*\*Generator annotáció és atrijaval; összetett azonosítók = több atriból áll össze: vagy entitásba több atri, rájuk @Id és osztályra @IdClass, vagy külön osztályban az atrik és rájuk @EmbeddedId és osztályra @Embeddable; csak egy objektumot lehet átadni paramként, azért kell így!)**

JPA-ban az entitásoknak **kötelező egyedi azonosítót** megadni; **ez legalább egy atri és az @Id** annotációval kell ellátni

lehetőség van **egyéb konfig paraméterek** megadására és arra is, hogy **összetett kulcsoka**t alkalmazzunk, amikor a kulcs több atriból áll

**Elsődleges kulcs típusa**

**elsődleges kulcsot** leggyakrabban akkor használjuk, **ha egy entitást egyedi azonosító alapján szeretnénk lekérdezni;** ilyenkor az **entitymanager.find()-**ot hívjuk, átadjuk első paraméterként az entitás osztályát, másodikként az egyedi azonosítót; **ált egész számokat használunk**; legegyszerűbb a használata (pl urles átadás webes alkalmazásnál)

**ne adjunk meg elsődleges kulcsként se doublet se idő alapú mezőt!** párhuzamosság probléma

**Azonosítógenerálás**

hogyan lehet az **azonosítógenerálást konfigurálni**?

**@GeneratedValue** annotáció és **stratégia** megadása; ha ez **AUTO**, akkor a **JPA implementációra bízzuk a választott algoritmust**(szekvencia, tábla vagy identity mező alapján történő azonosító generálás); **minden adatbázisban működni fog**



**Azonosítógenerálás táblával**

explicit módon ha meg szeretnénk adni, hogy **a generálás platformfüggetlen legyen**; ilyenkor válasszuk a táblát; **létrejött tábla tartalmazza, hogy a kiosztott azonosítók hol állnak**; ez biztosítja, hogy egy azonosító csak egyszer legyen kiosztva egy entitásnak

lehet ezt is **személyre szabni;** paraméterként megadjuk a generátor nevét és pluszba **@TableGenerator** annotáció és explicit módon még meg kell adni ezeket



**még további paraméterek megadása**, pl kezdeti érték (initial value; innen kezdi az azonosító kiosztását; allocaiton size: 100 jelenése: 100asával fordul az adatbázishoz az idkért, nem egyesével; lefoglalja az idkat 100asával a memóriában tartja és onnan osztja)



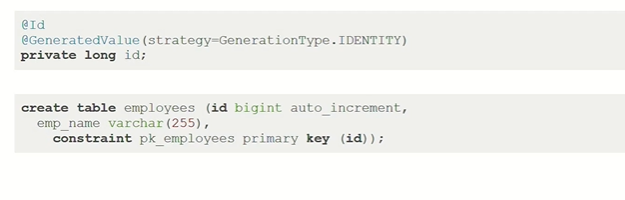
**Azonosítógenerálás szekvenciával**

ez már **adatbázisfüggő**, **van amelyik**ben nincs és **nem támogatja**, pl mysql nem

használat:****

**Azonosítógenerálás identity alapján – EZ A LÉNYEG**

ha azt szeretnénk, hogy az adatbázisban egy **oszlopnak a típusa legyen az**, hogy **automatikusan értéket adjon neki az adatbázis**, akkor ezt a paramétert kell átadni az annotációnak; az azonosítónak az adatbázis fog értéket adni



**Összetett azonosítók**

**több atriból álló; kétféleképpen** lehet

mindkettőnél **kell egy olyan osztály, ami tartalmazza a több atrit**; **mivel paraméterként egy paramétert adhatunk át**, ezért külön objektum erre az atrikkal

hogy az **entiás hogy reprezentálja az atrikat**, abba válaszhatunk:

1.**Entitásban az osztályban az azonosítós atrikra @Id és az osztályon @IdClass(idclass osztály neve)** annotáció és még külön IdClass osztály létrehozása, getset, equals, hash! Az IdClass osztálynak implementálnia kell a **Serializable** interfacet

2. **Az idclass osztályra @EmbeddedId**, az entitás osztályra pedig **@Embeddable,** és az **atri** közvetlen **beágyazása**

**Elsődleges kulcs – gyakorlat**

1. *id-n lévő @GeneratedValue() paramétert átírtuk ..AUTO-ra*

háttrében létrejön a *hibernate sequence tábla,* next\_val\_bigint oszloppal; a legközelebb kiadott entitás leendő idját tartalmazza(11); lockolja a kiadott sorszámot, és nem adható ki két entitásra; minden entitás várni fog a következő azonosítóra, párhuzamos felhasználás így nem támogatott?

1. *explicit módon hozunk létre a generálás alapjául szolgáló táblát; átírás .TABLE-re;* ilyenkor már *hibernate\_sequences tábla* generálódik, sequence\_name(default) és next\_val oszloppal (legutsó azonosító: 10)

*Tábla specifikálása:* (generator = „Emp\_Gen”), @TableGenerator(name=Emp\_Gen, table = emp\_id\_gen”, pkColumnName =”gen\_name”, valueColumnName = „gen\_val”) -> a gen\_val 100 volt, mert lefoglal valamennyi azonosítót előre, és ezeket membe tárolja, nem kell minden insertnél egy lock = select for update; *ez a megoldás platformfüggetlen és a sebességet is optimalizálni lehet vele*

1. *Szekvencia használata: strategy = . SEQUENCE*

vagy generator = „emp\_seq\_gen”), @SequenceGenerator(name =, sequenceName =””)…

1. *Összetett azonosító használata*

mindig egyesével meg fogom adni, de legyen összetett; pl egy String és egy szám:

a két atri mindkettő legyen az entitás része, legyenek getterek/setterek és *@Id* annotációk rajtuk és az osztályon *@Id..Class(EmployeId.class)* -> class létrehozása; **getter/setter és equals() hash() implementáció is kell!** metódusokban az id kicserélése az új employeId paraméterre; **EmployeeId még implementálja a Serializable-t;** de a depName új atri nem töltődik ki, explicit módon meg kell adni: új konstruktor és tesztek átírása

*másik fajta lehetőség: az EmployeeId osztályra @Embeddable*; Employee osztályból kiveszem az új atrikat és beteszem az EmployeeId atrit @EmbeddedId-val

**Kérdések**

*Mi kell minimum egy elsődleges kulcs megadásához?*

*Milyen típusai vannak az elsdleges kulcsnak?*

*Mit ne adjunk meg elsődleges kulcsként?*

*Milyen 3 típusa van az azonosítógenerálsának és milyne annótációk kellenek hozzá?*

*Hogy lehet összetett azonosítókat generálni?*