**Properties**

spring.jpa.hibernate.ddl-auto=

Aceasta proprietate permite lui Hibernate sa lucreze cu schema bazei de date

* **update** – compara schema bazei de date cu cea definita de noi si i da update la cea din baza de date, adica vede ce diferente sunt si o modifica, dar niciodata nu va sterge ceva din baza de date(coloana, tabel, constraint etc.), chiar daca e diferenta
* **create** – drop tabelele din baza de date si creaza altele pe baza la cele pe care noi le avem cu @Entity
* **validate** – verifica daca schemele coincid. Daca nu, arunca exceptie
* **create-drop** – face ceea ce face create si la final de program, sterge baza

**@Transient**

* Chiar daca nu punem anotatia @Column la un field, Hibernate oricum va crede ca acel field e o coloana din tabel. Daca nu vrem ca un field sa fie considerat ca colana, dar ca simplu field, folosim @Transient

**@Basic**

* Este exact ca @Column, dar permite sa decidem daca un field sa fie lazy sau eager initialized.

Basic(fetch=FetchType.LAZY)

Daca nu e specificat fetch, default e eager, si in asa caz nu e diferenta intre @Column si @Basic

**@Temporal**

* @Temporal ne permite sa lucram cu data, timp, adica cu DATE, TIME, TIMESTAMP din baza de date
* Lucrul cu datele si orele din Java in SQL e cam problematic, dar anotatia data ne ajuta

@Temporal(TemporalType.TIMESTAMP) – echivalent pentru TIMESTAMP din SQL

@Temporal(TemporalType.TIMES) – echivalent pentru TIME din SQL

@Temporal(TemporalType.DATE) – echivalent pentru DATE din SQL

* Aceasta anotatie e destinata mai mult fieldurilor ce sunt @Column

ManyToMany multiple fields

**Insertable and Updatable**

* La @Column sau @OneToOne etc. putem pune parametrii insertable=true/falsei si updatable=true/false
* **insertable=false** – inseamna ca Hibernate nu va include acest atribut cand entitatea va fi inserata in baza de date. Adica, pur si simplu nu va popula coloana si o va lasa cu null sau pe mana bazei de date, daca e id. In plus true e default
* **updatable=false** – Hibernate nu va da update la coloana data. Va da update la toate celelalte, dar asta va ramane asa cum a fost initial, indiferent ca noi o setam cu altceva prin setter.
* Sunt folosite mai mult pentru primaary key sau fk

**Caching**

* JpaRepository are 2 metode de save:

- save(Onject)

saveAndFlush(object)

Diferenta e ca cu save obiectul se va salva cand se va da commit la tranzactie, dar cu saveAndFlush, el se va salva deodata cand se executa metoda

* Acum, hibernate mereu pastreaza in cache rezultatele ce le ia din tabel, ca sa nu faca foarte multe interogari in baza de date. Totusi, daca de ex avem in BD 50 de clienti si hibernate i va stoca in cache, se poata intampla ca noi sa stergem din BD un client manual, accesand DBMS, insa Hibernate va pastra in cache inca 50 de clienti, desi noi deja am sters unul