**Advanced Mapping**

* One to one
* One to many, many to one
* Many to many

**Cascade**

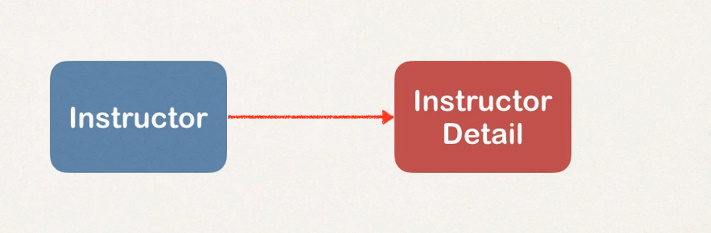
* Se refera la faptul ca orice operatie efectuata asupra unui tabel, se efectueaza si asupra celui cu care se leaga. De ex, daca avem tabelele Instructors si Info, si daca dam save la un instructor din tabel, el se va salva si in al doilea tabel, si daca il stergem din Instructors, el se va sterge si in Info.
* Acesta cu stersul e cunoscut ca CASCADE DELETE.

**Eager and Lazy Loading**

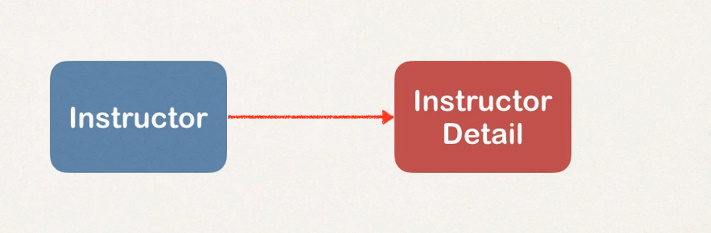
* Eager – retrieve everything. De ex, un instructor are mai mule cursuri, si obtinem deodata datele despre toate cursurile sale.
* Lazy – retrieve in request only. In acest caz, nu obtinem deodata toate datele despre cursurile sale, ci pe rand, treptat, dupa request.

**Uni-direction and Bi-direction relationshep**

* Uni-direction:



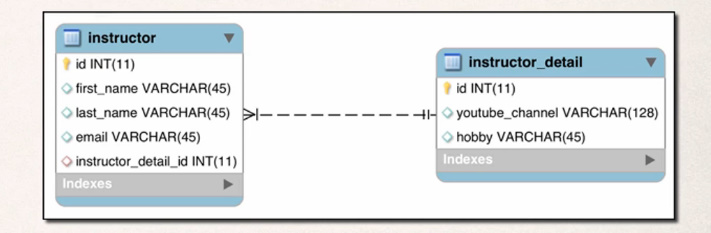
* Bi-direction





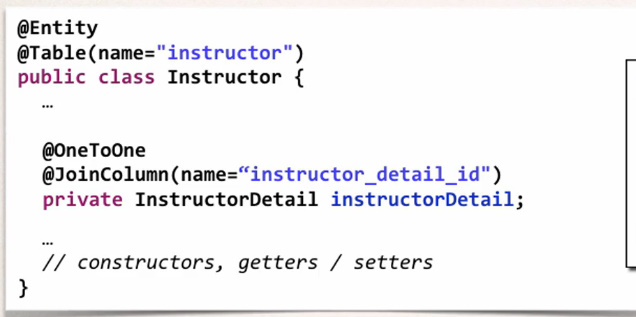
**@OneToOne**

Sa zicem ca avem 2 tabele:



Crearea claselor Entity pentru ambele e usoara, insa problema e ca in tabelul instructor exista o foreign key, si avem relatie de la unu la unu(presupnem). In tabele, in DBMS, putem usor arata asta, si trebuie cumva sa aratam asocierea asta si in clase.

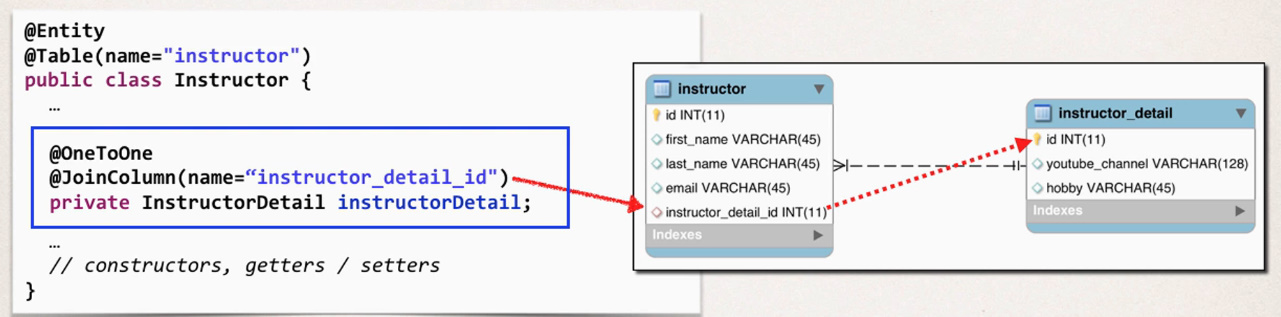
* Aici anume folosim anotatia @OneToOne



**@OneToOne** – relatia

**@JoinColumn(name=”atribut”)** – arata care e foreign key in Instructor, adica care atribut trebuie sa se lege de alta clasa mapata. „instructor\_detail\_id” nu trebuie definit in clasa, caci obiectul va comunica cu el prin intermediul la instructorDetail obeject, din el. In tabel, in baza de date, ar trebui sa scriem atributul care e foreign key, dar in clasa nu, e de ajuns obiectul la tabelul unde e el foreign key si numele coloanei lui, si ea va fi automat adaugata in clasa de sus.

**Private InstructorDetail instructorDetail** – spune lui Hibernate din care clasa e primary key cu care se leaga foreign key, si aici spunem ca anume din Clasa mapata cu numele InstructorDetail.



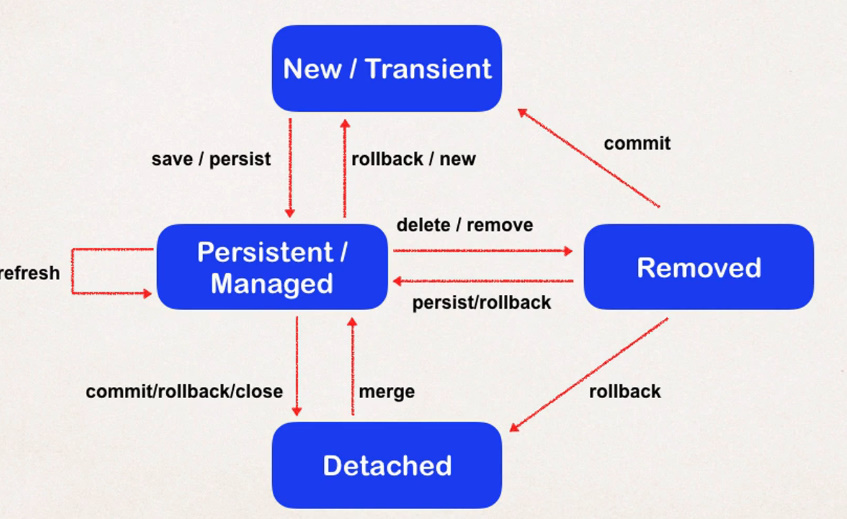
**Entity Life Cycle**

O stare prin care o hibernate entity poate trece sau in care se poate afla.

Operations+descriere:

* **Detach** – daca entitatea e detasata, ea nu este asociata cu nicio hibernate sesion
* **Merge** – daca o instanta este detasata de la sesie, atunci merge o va reatasa la sesie.
* **Persist** – transitioneaza noi instante spre managed state. Urmatorul comit o va salva in DB.
* **Remove** – tranzitioneaza entitatea managed spre stergere. Urmatorul commit o va sterge din baza de date.
* **Refresh** – reload obiecte cu date din DB. E folosit cand datele obiectelor din memorie difera de cele din DB.

**Entity Life Cycle Session method calls:**



**O entity se poate afla in starile:**

* **New/Transient** – un obiect de tip entitate nou creat.
* **Persistent/Managed** – stare in care obiectul apartine sesiunii, si deobicei, asupra obiectului s-au facut operatii, ca adaugarea de date,modificarea etc. si putem deja sa il salvam ori sterge. Putem sa-l ducem la starea anterioara. Daca vrem sa sincronizam obiectul cu datele din baza de date, facem refresh.
* **Detached** – odata ce dam commit, obiectul nu mai e asociat cu nicio sesiune Hibernate. Putem sa il reatasam la sesiunea Hibernate, trimitem referitnta la obiect si facem un mege asupra el.
* **Removed** – obiect sters. Putem sa il facem inapoi managed,sa il reintoarcem la transient dupa ce e sters din baza de date sau sa il facem din nou Detached..