* **Eager –** extragem toate datele
* **Lazy –** extragem date doar la request.

**Eager Loading**

Eager Loading extrage toate entitatile si dependentele lor. De ex, fie ca avem un tabel “teacher” si unul “course”. Fiecare course poate avea un profesor, dar un profesor poate avea mai multe cursuri. Aici, in entitati, e logic ca la Course vom avea foreign key catre teacher, insa teacher nu catre cursuri, caci nu ar avea cum. Totusi, teacher poate,daca vrem noi, avea un List<Course>, mapped by, si asa cu un profesor se va extrage o lista cu toate cursurile sale. De ex:

@Cascade(value = org.hibernate.annotations.CascadeType.*SAVE\_UPDATE*)  
@OneToMany(mappedBy = "teacher")  
private List<Course> courseList;

Totusi, problema e ca la fiecare selectie a unui profesor, mereu se va genera o lista cu toate cursurile sale. Uneori asta poate fi in plus, asa cum nu mereu avem nevoie de lista, si un Teacher nu trebuie neaprat sa o aiba. Hibernate a rezolvat problema data prin lazy loading.

**Eager Loading face** ca odata ce un profesor e extras, sa fie generata toata lista cursurilor sale. Asta poate incetini performanta aplicatiei.

**Lazy Loading**

* Lazy Loading rezolva problema de mai sus.
* Lazy Loading nu va genera automat lista pentru un Teacher, ci doar acel Teacher, si la **cerere**, si elementele din lista.
* **Lazy Loading e recomandat sa fie utilizat mereu.**

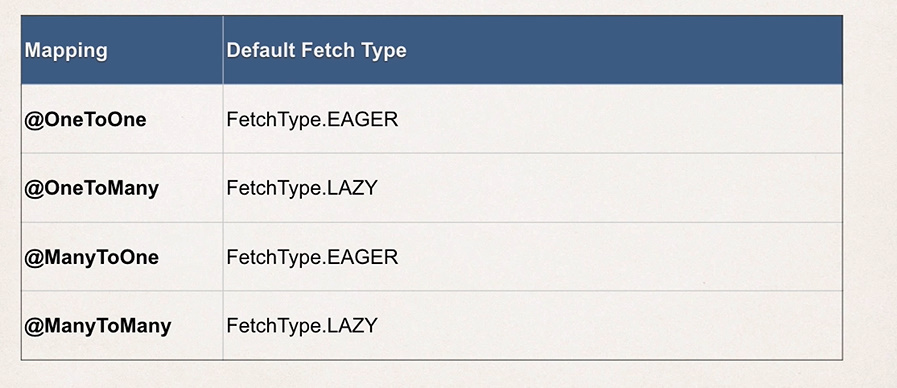
**fetch**

* Pentru a specifica cum anume vrem sa fie extrase aceste date, folosim

in **@OneToMany** sau altele, **fetch=FetchType.LAZY,** impreuna cu mappedBy

@Cascade(value = org.hibernate.annotations.CascadeType.*SAVE\_UPDATE*)  
@OneToMany(fetch = FetchType.*LAZY*,mappedBy = "teacher")  
private List<Course> courseList = new LinkedList<Course>();

* Totusi, fiecare tip de asociere are o valoare default pentru fetch:



* Totusi, chiar daca folosim FetchType.LAZY, asta nu inseamna ca obiectele dintr-o lista nu vor fi niciodata obtinute. De ex,

Daca Teacher Entity are o lista de cursuri si fetch=FetchType.LAZY:

@Cascade(value = org.hibernate.annotations.CascadeType.*SAVE\_UPDATE*)  
@OneToMany(fetch = FetchType.*LAZY*,mappedBy = "teacher")  
private List<Course> courseList = new LinkedList<Course>();

Si incercam sa obtinem acest Teacher, dar care are fetch=FetchType.LAZY,

Teacher teacher = session.get(Teacher.class,1);  
System.*out*.println(teacher);  
for(Course course : teacher.getCourseList())  
 System.*out*.println(course);

observam ca profesorul totusi are lista de cursuri, desi e LAZY. Ideea e ca in momentul in care teacher a fost returnat din baza de date, adica aici:

Teacher teacher = session.get(Teacher.class,1);

**el nu avea acea lista de cursuri**. Totusi, odata ce am apelat metoda getCourseList, care s-a atins de acea lista, Hibernate s-a asigurat sa ne returneze lista de cursuri din baza de date, ca sa nu fie null; Deci propriu zis, noi nu vedem modificari, dar vor fi la viteza programului.

**Session si Lazy Loading**

* **Daca avem nevoie de datele unui teacher, insa lista e Lazy, Hibernate va obtine lista din baza de date numai intr-o sesiune!**
* Daca deja am dat session.close() sau session.getTransaction().commit(), si vrem sa obtinem lista, se va arunca o exceptie

De ex:

@Cascade(value = org.hibernate.annotations.CascadeType.*SAVE\_UPDATE*)  
@OneToMany(fetch = FetchType.*LAZY*,mappedBy = "teacher")  
private List<Course> courseList = new LinkedList<Course>();

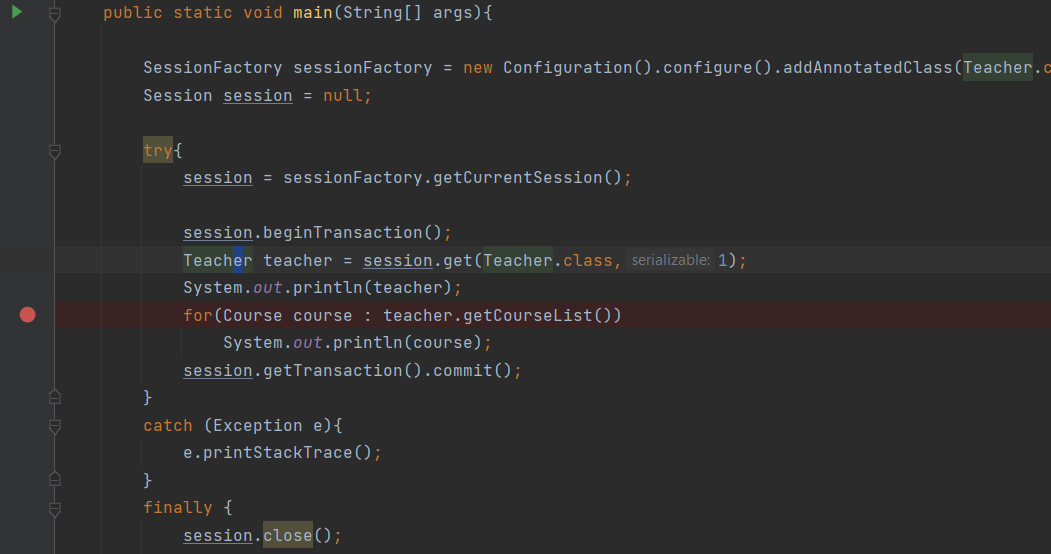
Daca vom face asa:

session = sessionFactory.getCurrentSession();  
  
session.beginTransaction();  
Teacher teacher = session.get(Teacher.class,1);  
System.*out*.println(teacher);  
  
session.getTransaction().commit();  
***session.close();***  
for(Course course : teacher.getCourseList())  
 System.*out*.println(course);

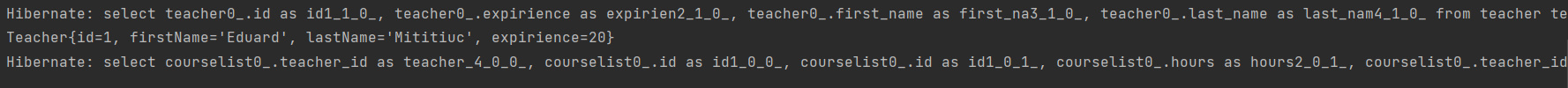
adica daca inchidem sesiunea sau dam commit, si apoi incercam sa accesam lista de cursuri, vom primi exceptie, asa cum lista e goala si Hibernate nu poate obtine obiectele din Baza de date. In cazul dat, avem @OneToMany, si fetch si asa e Lazy default, deci poate sa nu fie si pus, oricum se va intampla acelasi lucru.

* Putem pune breakpoint si rula in Debbug mode sa vedem asta. ATENTIE! Nu selectam lista de cursuri din Debug Mode sa vedem size, caci el oricum ne varata ce e in Proxy, adica desi lista e goala, va arata ce ar primi ea.

Facem asa:



Deci punem breakpoint la for, si se va ajunge pana la for. Dupa, daca mai dam un pas inainte, vom observa ca abea atunci in Consola apare cum Hibernate selecteaza din baza de date lista:





**Test2:**

try{  
 session = sessionFactory.getCurrentSession();  
  
 session.beginTransaction();  
 Course course = session.get(Course.class,25);  
  
 session.getTransaction().commit();  
 session.close();  
 System.*out*.println(course.getTeacher().getId());  
}



java.lang.NullPointerException: Cannot invoke "com.test.app.Course.getTeacher()" **because "course" is null**



at com.test.app.Hello.main(Hello.java:23)

**Lazy Issue Solve**

* Daca avem Lazy, gen cum la Teacher, lista de cursuri e Lazy, si daca in timpul sesiunii nu accesam lista, la inchiderea sesiunii ea va ramane goala.
* Accesarea listei nu e unica solutie de a ne asigura ca ea e setata.

**HQL JOIN FETCH**

* Daca lista e Lazy si nu vrem sa o accesam cand e deschisa sesiunea pentru a o primi, putem folosi o interogare cu JOIN FETCH
* Query query = session.createQuery("FROM Teacher t JOIN FETCH t.courseList WHERE t.id=:id").setParameter("id",1);

JOIN FETCH t.courseList va face ca efectul lui LAZY sa fie anulat pentru courseList din query, si oricum sa se returneze lista, chiar de e setata ca LAZY.

session = sessionFactory.getCurrentSession();  
  
 session.beginTransaction();  
Query query = session.createQuery("FROM Teacher t JOIN FETCH t.courseList WHERE t.id=:id").setParameter("id",1);  
Teacher teacher = (Teacher) query.getSingleResult();  
 session.getTransaction().commit();  
 session.close();  
System.*out*.println(teacher.getCourseList());



**RESOLVE FETCH ISSUE AFTER CLOSE SESSION**

* Totusi, daca peste un timp, dupa ce sesiunea a fost inchisa, ne dam seama ca avem nevoie cumva sa umplem lista goala, o facem pur si simplu prin obtinereaa listei din baza de date cu query si o setam la teacher.

session = sessionFactory.getCurrentSession();  
  
 session.beginTransaction();  
 Teacher teacher = session.get(Teacher.class,1);  
 session.getTransaction().commit();  
 session.close();  
  
  
  
 session = sessionFactory.getCurrentSession();  
 session.beginTransaction();  
 List<Course> courseList = session.createQuery("FROM Course c WHERE c.teacher.id = :ID").setParameter("ID",1).getResultList();  
 teacher.setCourseList(courseList);  
 System.*out*.println(teacher.getCourseList());  
 session.close();  
}

**Interogarea nu va crea aici obiecte, adica c.teacher.id nu inseamna ca se creaza obiect sau ceva chiar la rularea interogarii. Deci c.teacher.id doar ii indica la ce fielduri sa se uite din clasa Entity, ca se la gaseasca anotatia @Column si sa vada ce campuri le corespund in tabelul bazei de date.**