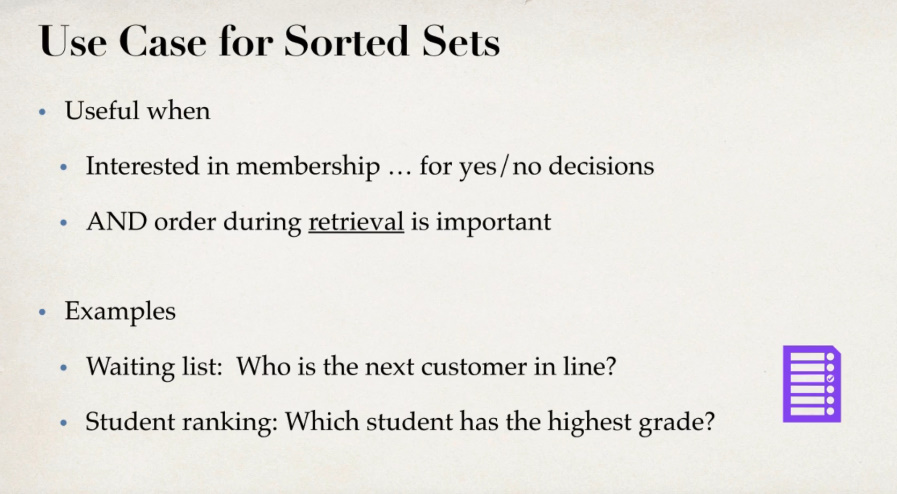
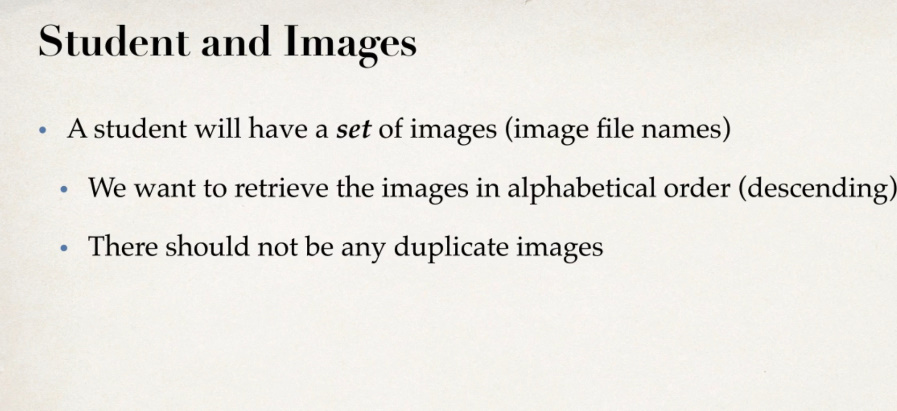
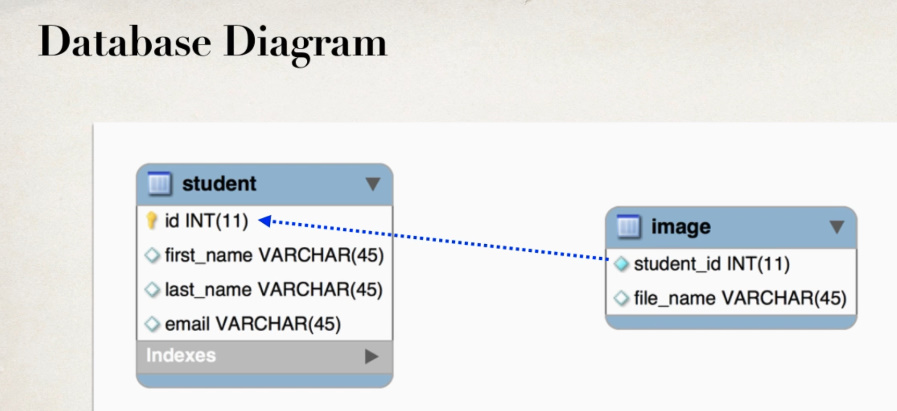
**Sorted Set**



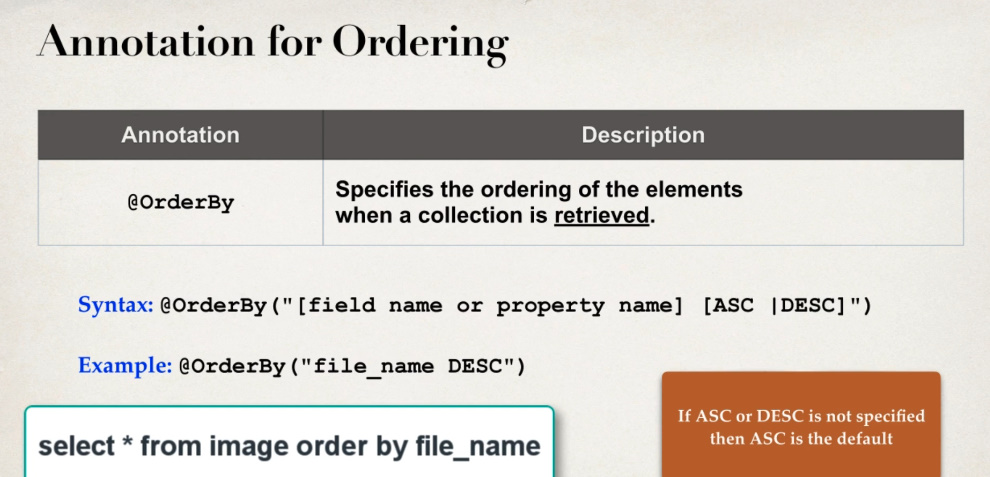


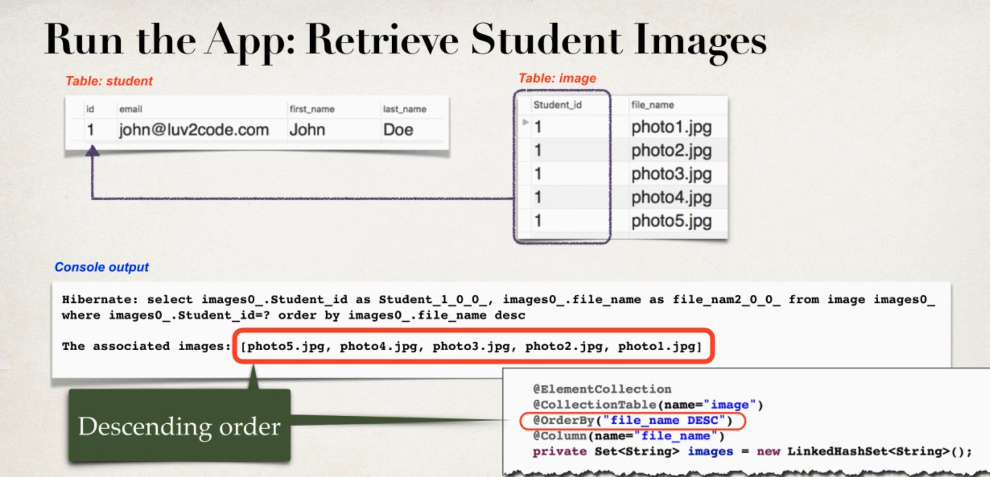




**@OrderBy(“column ASC|DESC”)**

* Deci, specificam coloana dupa care sa se faca sortarea



* Coloana pusa in @OrderBy si @ColumnName pot fi diferite. Asta doar influenteaza cum datele vor fi extrase din baza de date, si pe baza la asta, se va memora ordinea cum ele vin
* @OrderBy are foar grija ca atunci cand Hibernate scoate datele din baza de date, sa mai introduca la Query si ORDER BY ....
* 
* @ElementCollection  
  @CollectionTable(name = "image",joinColumns = @JoinColumn(name = "student\_id"))  
  @Column(name = "file\_name")  
  @OrderBy("file\_name ASC")  
  private Set<String> images = new HashSet<>();

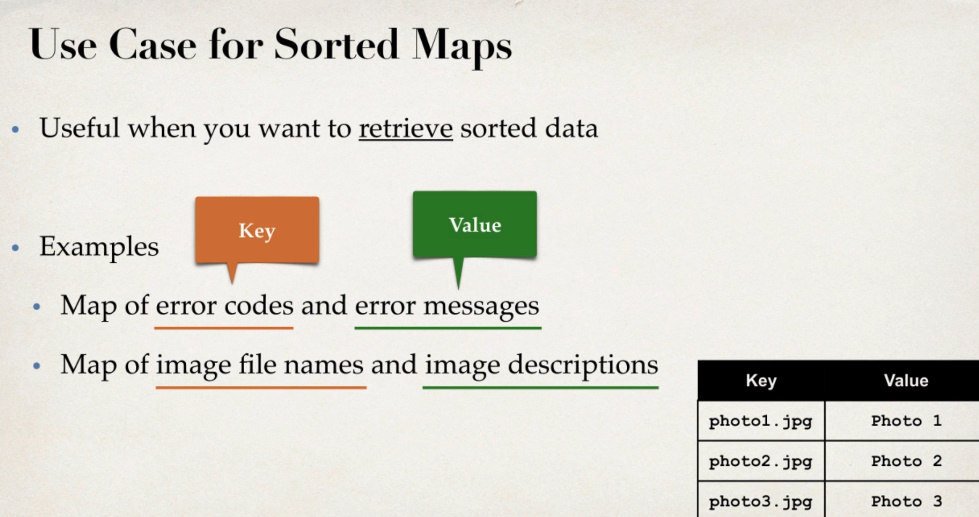
Putem si sa nu punem atributul joinColumns in @CollectionTable, caci oricum se va cauta un atribut in tabel cu numele student\_id, asa cum @Tabel nume la Student e “student” si automat va fi cautata student\_id, dar mai bine specificam totusi

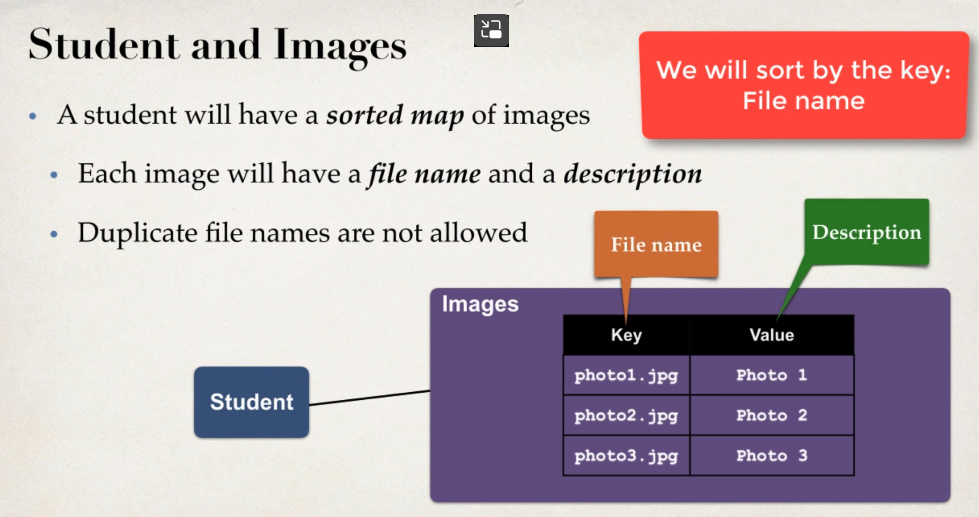
* **Este important sa folosim un LinkedHashSet**
* **Daca nu folosim joinColumns, automat se va lua numele la entity curent + \_id, gen student\_id, dar daca asa column nu exista, avem problema, de aceea mai bine scriiem.**
* Anotatia data poate fi folosita si fara a avea @ElementCollection.
* @OrderBy de la JPA nu permite sortarea dupa mai multe coloane. De la Hibernate permite, folosind clause = “” si mai multe scriem
* Daca numele la coloana pusa in @Column si @OrderBy e acelasi, putem poate doar @OrderBy si fata numele la coloana:
* @ElementCollection  
  @CollectionTable(name = "image",joinColumns = @JoinColumn(name = "student\_id"))  
  @Column(name = "file\_name")  
  @OrderBy  
  private Set<String> images = new HashSet<>();

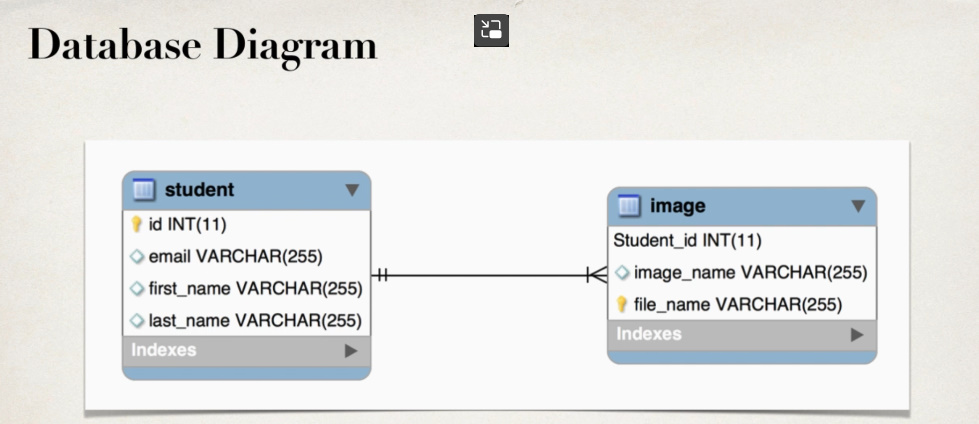
**@OrderBy vs @OrderColumn**

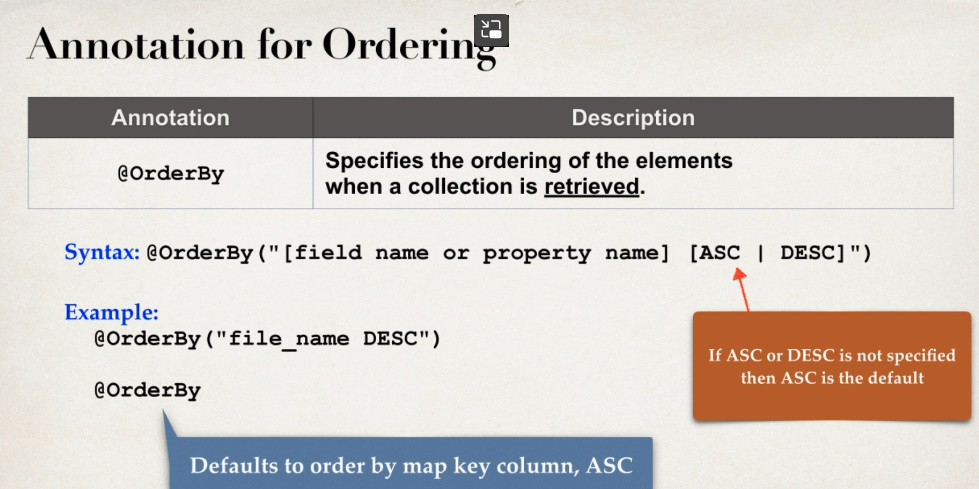
* @OrderColumn e folosit cand avem o coloana speciala pentru a memora ordinea de adaugare in baza de date.
* @OrderBy se pune pentru a sorta datele dupa orice coloana, care nu este o coloana pentru a pastra ordinea de insertie

**Sorted Maps**



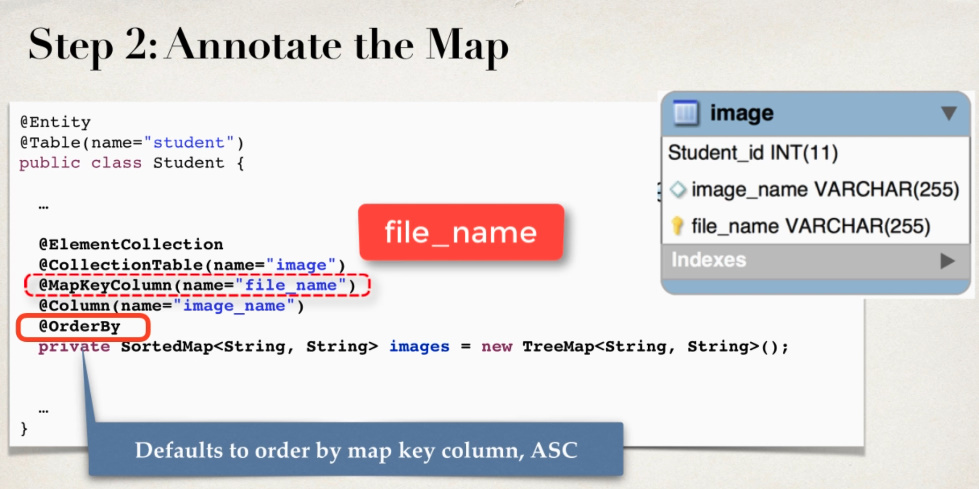






deci, nu e nici-i diferenta fata de order la Sets.

* Vedem ca daca punem doar @OderBy, sortarea se va face automat ASC dupa key, DAR UNEORI POATE SA NU MEARGA. MAI BINE SCRIEM NUMELE LA COLOANA KEY





**Folosim Map, nu SortedMap, si folosim TreeMap sau LinkedHashMap**

**@ElementCollection**@CollectionTable(name = "image",joinColumns = @JoinColumn(name = "student\_id"))  
@MapKeyColumn(name = "file\_name")  
@Column(name = "image\_name")  
@OrderBy("file\_name ASC")  
private Map<String,String> images = new LinkedHashMap<>();

**@SortComparator(Clasa.class)**

Anotatia data ne permite sa definim comparatorul care sa sorteze colectia. In aczul dat, Map va fi sortat logic ca dupa Key

@ElementCollection  
@CollectionTable(name = "image",joinColumns = @JoinColumn(name = "student\_id"))  
@MapKeyColumn(name = "file\_name")  
@Column(name = "image\_name")  
@SortComparator(Compare.class)  
private Map<String,String> images = new LinkedHashMap<>();  
  
public static class Compare implements Comparator<String>{  
  
 @Override  
 public int compare(String o1, String o2) {  
 return o1.compareTo(o2);  
 }  
}