CS544 EA

Hibernate

JPQL: SELECT clause

Select Clause

A result can contain:
Entities
Properties
A mix of the two

- SELECT specifies **which entities or properties** the query should return
  - We've already seen selecting an entity

```
TypedQuery<Person> q = em.createQuery("select distinct p from Person p "
          + "left outer join p.address a ", Person.class);
```

  - It's easy to select a single property

```
TypedQuery<String> q = em.createQuery("select p.firstName from Person p ", String.class);
```

Selects the firstName of all Person Objects

Multiple Items

- You can specify more than one entity / property
  - By default these are **returned as an `Object[]`**

```
TypedQuery<Object[]> q = em.createQuery(
          "select person, pet.species, adr.city "
          + "from Pet pet join pet.owner person, "
          + "join person.address adr", Object[].class);
List<Object[]> result = q.getResultList();

Person p = null; String petType = null; String city = null;
for (Object[] item : result) {
  p = (Person) item[0]; petType = (String) item[1]; city = (String) item[2];

  System.out.println(p.getFirstName() + " " + p.getLastName()
        + " owns a " + petType + " in " + city);
}
```

List

- Use **`new list()`** in JPQL to select as List

```
Query q = em.createQuery(
          "select new list(person, pet.species, adr.city) "
          + "from Pet pet join pet.owner person, "
          + "join person.address adr");
List<List<Object>> result = q.getResultList();


Person p = null; String petType = null; String city = null;
for (List<Object> item : result) {
     p = (Person) item.get(0);
     petType = (String) item.get(1);
     city = (String) item.get(2);

     System.out.println(p.getFirstName() + " " + p.getLastName()
        + " owns a " + petType + " in " + city);
}
```

I have not been able to make this work with a TypedQuery

Map

- **new Map()** selects as Map<String, Object>
  - Requires you to give aliases to each element
  - Alias will be used as Key in the map

```
Query query = em.createQuery(
  "select new map(p as person, sum(a.balance) as liquid) "
  + "from Person p join p.accounts a group by p.id ");

List<Map<String,Object>> items = query.getResultList();

Person p = null; Double liquid = null;
for (Map<String,Object> item : items) {
  p = (Person)item.get("person");
  liquid = (Double)item.get("liquid");

  System.out.println(p.getFirstName() + " " + p.getLastName()
      + "'s liquid assets: " + liquid);
}
```

New Object

- Results can be of **any object**
  - Do need constructor for what you provide
  - Ideal for constructing DTOs

```
TypedQuery<Home> query = em.createQuery(
     "select new hibernate06.Home(p, a) "
     + "from Person p " + "join p.address a ", Home.class);
List<Home> homes = query.getResultList();

Person p = null;
Address a = null;
for (Home home : homes) {
     p = home.getPerson();
     a = home.getAddress();

     System.out.println(p.getFirstName()
     + " " + p.getLastName()
     + " has a home in " + a.getCity());
```

Needs fully-qualified class name

No need for Casting!

Not an Entity just some class

```
public class Home {
     private Person person;
     private Address address;

     public Home(Person p, Address a) {
          this.person = p;
          this.address = a;
     }
```

Aggregates

- JPQL also has the typical **aggregate** functions
  - avg(…), sum(…), min(…), max(…)
  - count(*), count(…), count(distinct …), count(all …)

```
Query query = em.createQuery(
     "select new map(p as person, sum(a.balance) as liquid) "
     + "from Person p join p.accounts a group by p.id "
     + "having liquid > 100 ");
```

Sum of the balance of all acounts related to one Person and having at least 100

- **Group By** clause specifies groups to aggregate
- The **having** clause can filter groups