CS544 EA

Hibernate

JPQL: FROM clause

FROM Clause

- Simplest query only contains a FROM clause
  - The SELECT clause is optional (unlike SQL)

```
TypedQuery<Person> query1 = em.createQuery("from Person", Person.class);
```

- JPQL **keywords** are not case sensitive
  - FROM, from, FrOm, fRoM are all the same
- **Class and property** names are case sensitive
  - Person and person are two different classes!

Entity names, Property names

- JPQL Queries are based on Java:
  - Always use the **Entity** (class) name
  - Always use the **property** name

- JPQL never uses table or column names

Reminder: @Entity and @Table

- @Entity(name=”OtherName”)
  - Changes table name
  - Changes **what the entity is called** (in a query)

- @Table(name=”othername”)
  - Only changes the table name
  - Inside a query the class name is used

Polymorphic Queries

- JPQL has **excellent polymorphism** support
  - Returns all Accounts regardless of sub-type

```
TypedQuery<Account> query = em.createQuery("from Account", Account.class);
List<Account> accounts = query.getResultList();
```

  - Returns all objects that implement the interface

```
TypedQuery<Serializable> query2 = em.createQuery("from Serializable", Serializable.class);
List<Serializable> serializables = query2.getResultList();
```

  - Returns every (entity) object in the database!

```
TypedQuery<Object> query3 = em.createQuery("from Object", Object.class);
List<Object> objects = query3.getResultList();
```

Aliases

- Alias entity names for ease of reference
  - Just like SQL aliases for table names

```
TypedQuery<Person> query1 = em.createQuery("from Person as p", Person.class);
```

  - Just like SQL the ‘as’ keyword is optional

```
TypedQuery<Person> query1 = em.createQuery("from Person p", Person.class);
```

Pagination

- JQPL has built-in pagination support
  - Selects a **part of a bigger result set**
  - Does not necessarily speed things up

```
em.getTransaction().begin();
TypedQuery<Person> pplQuery = em.createQuery("from Person", Person.class);

pplQuery.setFirstResult(0);
pplQuery.setMaxResults(50);

List<Person> ppl = pplQuery.getResultList();
for (Person p : ppl) {
     System.out.println(p);
}
em.getTransaction().commit();
```

Zero based index - 0 is first item

Page size

Returns first 50 people

Order By

- The ‘order by’ clause **sorts the result** by that

```
TypedQuery<Person> query1 = em.createQuery("from Person p order by p.lastName", Person.class);
```

By default sorted ascending (ASC)

- The **ASC or DESC keywords** can be added to sort in ascending or descending order

```
TypedQuery<Person> query1
     = em.createQuery("from Person p order by p.lastName DESC", Person.class);
```

Sorted Descending