CS544 EA

Hibernate

JPQL: Joins

Joins

- There are 2 types of joins:
  - Explicit joins that use the JOIN keyword

```
TypedQuery<Person> q = em.createQuery("select p from Person as p  "
                    + " JOIN p.address as a where a.city = 'Fairfield'", Person.class);
```

  - Implicit joins that don’t use JOIN
    - Instead follow references by using the . operator

```
TypedQuery<Person> q = em.createQuery("from Person as p "
                    + " where p.address.city = 'Fairfield'", Person.class);
```

Explicit Joins

- Syntax for explicit join is:
  - JOIN table.property [as] alias
  - Alias can then be used inside WHERE clause

```
TypedQuery<Person> q = em.createQuery("select p from Person as p  "
                    + " JOIN p.address as a where a.city = 'Fairfield'", Person.class);
```

- Explicit join expands the result set
  - Have to **use a SELECT** clause to bring it back to one entity

Implicit Joins

- Implicit follows reference
  - **Only works for references**
  - Does not expand result set (no need for select)

@OneToOne
@ManyToOne

```
TypedQuery<Person> q = em.createQuery("from Person as p "
                    + " where p.address.city = 'Fairfield'", Person.class);
```

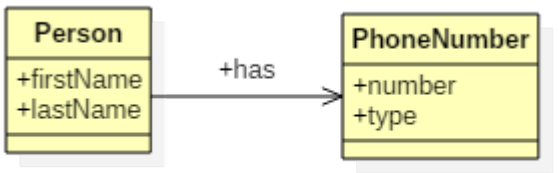
- You cannot implicit join a collection
  - Using [ ] with an indexed collection turns the collection into a reference

@OneToMany
@ManyToMany

Joining a Collection

- Joining a collection requires:
  - **Explicit join**, therefore also a **Select clause**
  - And the **Distinct keyword**

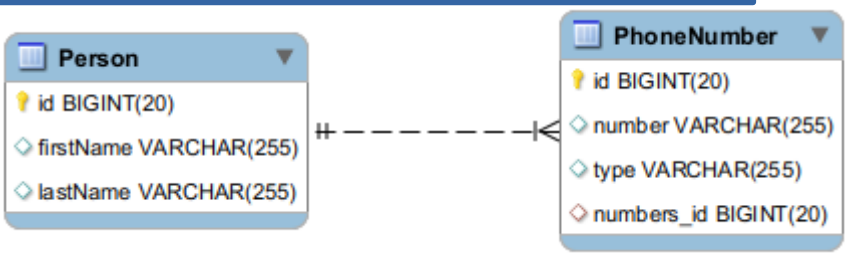
- First an example with just the explicit join
  - No Select
  - No Distinct

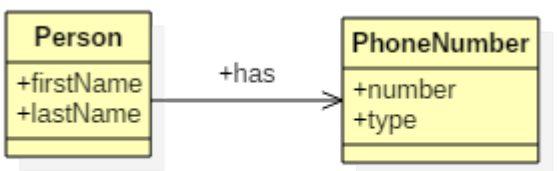


Mapped Domain

```java
@Entity
public class Person {
     @Id
     @GeneratedValue
     private Long id;
     private String firstName;
     private String lastName;
     @OneToMany
     @JoinColumn
     private List<PhoneNumber> numbers
          = new ArrayList<>();
```

```java
@Entity
public class PhoneNumber {
     @Id
     @GeneratedValue
     private Long id;
     private String number;
     private String type;
```





id	firstName	lastName
1	Edward	Towers
2	John	Brown

id	number	type	numbers_id
1	641-472-1234	Home	1
2	641-919-5432	Mobile	1
3	641-233-9876	Mobile	2
4	641-469-4567	Home	2

Joining a Collection Without Select or Distinct

```
TypedQuery<Object[]> q = em.createQuery("from Person p "
          + "join p.numbers as n "
          + "where n.number like '641%'", Object[].class);
```

- **Without Select** result contains **2 entities**

Person	PhoneNumber
Edward, Towers	home, 641-472-1234
Edward, Towers	mobile, 641-919-5432
John, Brown	mobile, 641-233-9876
John, Brown	home, 641 469-4567

Joining Collection Without Distinct

```
TypedQuery<Person> q = em.createQuery("select p from Person p "
          + "join p.numbers as n "
          + "where n.number like '641%'", Person.class);
```

- The select gives us a single entity
  - **Still have duplicates** because of join!

Person
Edward, Towers
Edward, Towers
John, Brown
John, Brown

Distinct

- Distinct **removes duplicate** rows

```
TypedQuery<Person> q = em.createQuery("select distinct p from Person p "
          + "join p.numbers as n "
          + "where n.number like '641%'", Person.class);
```

Person
Edward, Towers
John, Brown

- Joining a collection therefore requires:
  - Explicit **Join** with a **Select**
  - And the **Distinct** keyword

Inner Joins

- All joins so far have been inner joins
  - If **one side is null there is no join**, no result row
- Outer Joins are also possible
  - Allow one of the sides to be null
  - Includes data that could not join

Person	Address
Edward, Towers	New York, New York
John, Brown	
Alice, Doe	Los Angeles, California

John Brown included even though no Address

Left Outer Join

- Left Outer Join means:
  - Data on the left (where we start) has to be there
  - Data that is joined (on the right) can be null

Right Outer Join
Not supported by JPA

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

Do not need both
"left" and "outer"

Can say: left join
Can say: outer join

Person	Address
Edward, Towers	New York, New York
John, Brown	
Alice, Doe	Los Angeles, California

Join Fetch

- Join Fetch lets you Join so that:
  - Related entities are added to the EM Cache
  - Without adding them to the resultset

- Useful for avoiding the N+1 problem
  - We will talk about this more during optimization

Join Fetch

No SELECT clause needed
joined entities not added to ResultSet

```
TypedQuery<Customer> query = em.createQuery(
          "from Customer c "
          + "JOIN FETCH c.address a "
          + "JOIN FETCH c.books b "
          + "JOIN FETCH b.author "
          + "WHERE c.firstName like :name",
Customer.class);
query.setParameter("name", "J%");
List<Customer> customers = query.getResultList();
System.out.println(customers.size());
```

Important: don't join (fetch)
multiple collections!

This creates a Cartesian Product
(see optimization)

```
Hibernate:
    select
        customer0_.id as id1_3_0_,
        address1_.id as id1_0_1_,
        books2_.id as id1_2_2_,
        author3_.id as id1_1_3_,
        customer0_.address_id as address_4_3_0_,
        customer0_.firstName as firstNam2_3_0_,
        customer0_.lastName as lastName3_3_0_,
        address1_.city as city2_0_1_,
        address1_.state as state3_0_1_,
        books2_.author_id as author_i3_2_2_,
        books2_.name as name2_2_2_,
        books2_.books_id as books_id4_2_0__,
        books2_.id as id1_2_0__,
        author3_.name as name2_1_3_
    from
        Customer customer0_
    inner join
        Address address1_
            on customer0_.address_id=address1_.id
    inner join
        Book books2_
            on customer0_.id=books2_.books_id
    inner join
        Author author3_
            on books2_.author_id=author3_.id
    where
        customer0_.firstName like ?
1
```