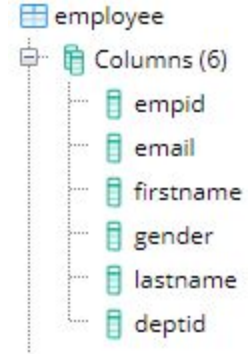
Criteria in Hibernate

I. Do we need Criteria:

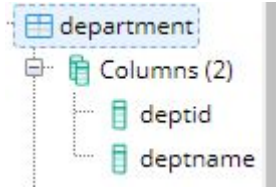
- ***Advantages:***
  - ***Criteria is really good at handling many optional search parameters:***
    - Allows you to build up a criteria query object programmatically where you can apply filtration rules and logical conditions.
    - Avoid using StringBuilder in HQL.
  - Easy to read, debug ?
  - Avoid syntax errors in HQL.

- **Disadvantages:**
  - Complex queries written in HQL are easier to read.
  - Combine with native sql because of lack of some functions.

```java
public class EmployeeEntity {

    @Column(name="empId")
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;
    private String firstName;
    private String lastName;
    private String gender;

    @NotNull
    @Email
    private String email;

    @ManyToOne
    @JoinColumn(name="deptid")
    private DepartmentEntity department;
}
```




```java
public class DepartmentEntity {
    @Column(name="deptId" )
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;

    @Column(name="deptname")
    private String deptName;

    @OneToMany(mappedBy = "department")
    List<EmployeeEntity> employeeEntities;
}
```



II. Basic criteria queries:

How to get all employees ?

- HQL: from EmployeeEntity
- Native sql: select * from employee
- session.createCriteria(EmployeeEntity.class)
  .list();

ID	First name	Last name	Gender	Email
2	Quan 1	Tang Kien	female	quan.tang.1@axonactive.com
1	Phuc	Dang Hoang	male	phuc.dang@axonactive.com
9	Else 2	Someone	female	someone.else.2@axonactive.com
8	Else 1	Someone	female	someone.else.1@axonactive.com
7	Vollmer 1	Kai	undefine	kai.vo.1@axonactive.com
5	Quan 2	Tang Kien	male	quan.tang.2@axonactive.com
3	Vollmer 2	Kai	undefine	kai.vo.2@axonactive.com
6	Vollmer 3	Kai	male	kai.vo.3@axonactive.com
4	Quan 3	Tang Kien	female	quang.tang.3@axonactive.com
10	Else 3	Someone	female	someone.else.3@axonactive.com

1. Narrowing the result set:

How to get all female employees?

```
select * from employee
where gender = 'female'
```

```
session.createCriteria(EmployeeEntity.class)
 .add(Restrictions.eq("gender", "female"))
 .list();
```

ID	First name	Last name	Gender	Email
2	Quan 1	Tang Kien	female	quan.tang.1@axonactive.com
4	Quan 3	Tang Kien	female	quang.tang.3@axonactive.com
8	Else 1	Someone	female	someone.else.1@axonactive.com
9	Else 2	Someone	female	someone.else.2@axonactive.com
10	Else 3	Someone	female	someone.else.3@axonactive.com

1. Narrowing the result set:

How to get all female employees having last name is Tang Kien?

```sql
select * from employee
where gender = 'female'
and lastname = 'Tang Kien'
```

```
session.createCriteria(EmployeeEntity.class)
 .add(Restrictions.eq("gender", "female"))
 .add(Restrictions.eq("lastName", "Tang Kien"))
 .list();
```

ID	First name	Last name	Gender	Email
2	Quan 1	Tang Kien	female	quan.tang.1@axonactive.com
4	Quan 3	Tang Kien	female	quang.tang.3@axonactive.com

1. Narrowing the result set:

Other Restrictions

- gt : greater than
- lt : less than
- like
- between
- isNull
- isNotNull
- in
- ...

1. Narrowing the result set:

SqlRestriction

```
session.createCriteria(EmployeeEntity.class)
 .add(Restrictions.sqlRestriction("length({alias}.firstname) > ?", 8, IntegerType.INSTANCE))
 .list();
```

```
select * from employee e where length(e.firstname) > 8
```

ID	First name	Last name	Gender	Email
7	Vollmer 1	Kai	undefine	kai.vo.1@axonactive.com
3	Vollmer 2	Kai	undefine	kai.vo.2@axonactive.com
6	Vollmer 3	Kai	male	kai.vo.3@axonactive.com

2. Ordering the results:

Order.asc-Order.desc

```
select * from employee
order by lastname asc
```

```
session.createCriteria(EmployeeEntity.class)
 .addOrder(Order.asc("lastName"))
 .list();
```

ID	First name	Last name	Gender	Email
1	Phuc	Dang Hoang	male	phuc.dang@axonactive.com
7	Vollmer 1	Kai	undefine	kai.vo.1@axonactive.com
3	Vollmer 2	Kai	undefine	kai.vo.2@axonactive.com
6	Vollmer 3	Kai	male	kai.vo.3@axonactive.com
8	Else 1	Someone	female	someone.else.1@axonactive.com
9	Else 2	Someone	female	someone.else.2@axonactive.com
10	Else 3	Someone	female	someone.else.3@axonactive.com
5	Quan 2	Tang Kien	male	quan.tang.2@axonactive.com
2	Quan 1	Tang Kien	female	quan.tang.1@axonactive.com
4	Quan 3	Tang Kien	female	quang.tang.3@axonactive.com

2. Ordering the results:

Order.asc-Order.desc

```
select * from employee
order by lastname asc,
firstname asc
```

```
session.createCriteria(EmployeeEntity.class)
 .addOrder(Order.asc("lastName"))
 .addOrder(Order.asc("firstName"))
 .list();
```

ID	First name	Last name	Gender	Email
1	Phuc	Dang Hoang	male	phuc.dang@axonactive.com
7	Vollmer 1	Kai	undefine	kai.vo.1@axonactive.com
3	Vollmer 2	Kai	undefine	kai.vo.2@axonactive.com
6	Vollmer 3	Kai	male	kai.vo.3@axonactive.com
8	Else 1	Someone	female	someone.else.1@axonactive.com
9	Else 2	Someone	female	someone.else.2@axonactive.com
10	Else 3	Someone	female	someone.else.3@axonactive.com
2	Quan 1	Tang Kien	female	quan.tang.1@axonactive.com
5	Quan 2	Tang Kien	male	quan.tang.2@axonactive.com
4	Quan 3	Tang Kien	female	quang.tang.3@axonactive.com

3. Combining expressions with logical operators:

```
select * from employee
where firstname like 'P%'
or firstname like 'Q%'
```

or

```
session.createCriteria(EmployeeEntity.class)
 .add(Restrictions.or(
        Restrictions.like("firstName", "P%"),
        Restrictions.like("firstName", "Q%")))
 .list();
```

ID	First name	Last name	Gender	Email
1	Phuc	Dang Hoang	male	phuc.dang@axonactive.com
2	Quan 1	Tang Kien	female	quan.tang.1@axonactive.com
4	Quan 3	Tang Kien	female	quang.tang.3@axonactive.com
5	Quan 2	Tang Kien	male	quan.tang.2@axonactive.com

3. Combining expressions with logical operators:

Combine and (conjunction), or

```
select * from employee
where firstname like 'P%' and gender = 'male'
or firstname like 'Q%' and gender = 'female'
```

```
session.createCriteria(EmployeeEntity.class)
 .add(Restrictions.or(
        Restrictions.and(
                Restrictions.like("firstName", "P%"),
                Restrictions.eq("gender", "male")
        ),
        Restrictions.and(
                Restrictions.like("firstName", "Q%"),
                Restrictions.eq("gender", "female")
                )
        ))
 .list();
```

4. Subqueries:

Get all departments having at least 2 female employees?

```
select * from department d where
2 <= (select count(e.empId)
      from employee e
      where e.deptid=d.deptId
      and e.gender='female')
```

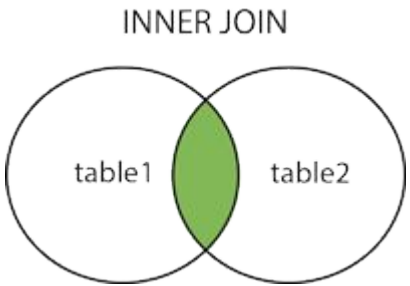
ID	Department Name
2	Information Technology

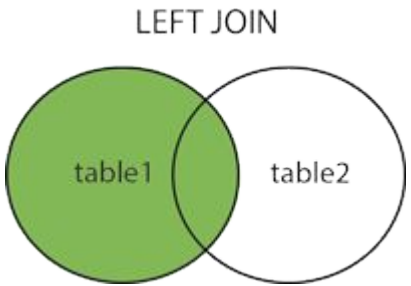
```
DetachedCriteria subquery = DetachedCriteria
                                .forClass(EmployeeEntity.class, "e");
subquery.add(Restrictions.eqProperty("e.department.id", "d.id"))
        .add(Restrictions.eq("gender", "female"))
        .setProjection(Property.forName("e.id").count());
return session.createCriteria(DepartmentEntity.class, "d")
              .add(Subqueries.le(2l, subquery))
              .list();
```

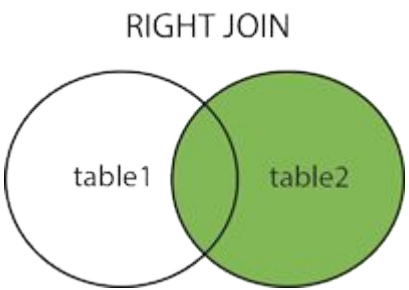
III. Joins and dynamic fetching:

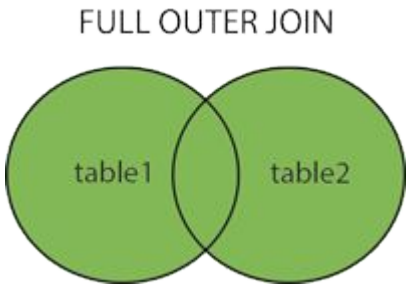
How many types of join ?

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table









III. Joins and dynamic fetching:

Inner join

```
select * from department d inner join employee e on d.deptId = e.deptid
```

deptid integer	deptname character varying (255)	empid integer	email character varying (255)	firstname character varying (255)	gender character varying (255)	lastname character varying (255)	deptid integer
1	Developer	2	quan.tang.1@axonactive.c...	Quan 1	female	Tang Kien	1
1	Developer	1	phuc.dang@axonactive.com	Phuc	male	Dang Hoang	1
2	Information Technology	9	someone.else.2@axonactiv...	Else 2	female	Someone	2
2	Information Technology	8	someone.else.1@axonactiv...	Else 1	female	Someone	2
2	Information Technology	7	kai.vo.1@axonactive.com	Vollmer 1	undefine	Kai	2
2	Information Technology	5	quan.tang.2@axonactive.c...	Quan 2	male	Tang Kien	2
2	Information Technology	3	kai.vo.2@axonactive.com	Vollmer 2	undefine	Kai	2
3	Finance	6	kai.vo.3@axonactive.com	Vollmer 3	male	Kai	3
3	Finance	4	quang.tang.3@axonactive....	Quan 3	female	Tang Kien	3

III. Joins and dynamic fetching:

Left outer join

```
select * from department d left outer join employee e on d.deptId = e.deptid
```

deptid integer	deptname character varying (255)	empid integer	email character varying (255)	firstname character varying (255)	gender character varying (255)	lastname character varying (255)	deptid integer
1	Developer	2	quan.tang.1@axonactive.c...	Quan 1	female	Tang Kien	1
1	Developer	1	phuc.dang@axonactive.com	Phuc	male	Dang Hoang	1
2	Information Technology	9	someone.else.2@axonactiv...	Else 2	female	Someone	2
2	Information Technology	8	someone.else.1@axonactiv...	Else 1	female	Someone	2
2	Information Technology	7	kai.vo.1@axonactive.com	Vollmer 1	undefine	Kai	2
2	Information Technology	5	quan.tang.2@axonactive.c...	Quan 2	male	Tang Kien	2
2	Information Technology	3	kai.vo.2@axonactive.com	Vollmer 2	undefine	Kai	2
3	Finance	6	kai.vo.3@axonactive.com	Vollmer 3	male	Kai	3
3	Finance	4	quang.tang.3@axonactive....	Quan 3	female	Tang Kien	3
4	Human Resources	[null]	[null]	[null]	[null]	[null]	[null]

III. Joins and dynamic fetching:

Right outer join

```
select * from department d right outer join employee e on d.deptId = e.deptid
```

deptid integer	deptname character varying (255)	empid integer	email character varying (255)	firstname character varying (255)	gender character varying (255)	lastname character varying (255)	deptid integer
1	Developer	2	quan.tang.1@axonactive.c...	Quan 1	female	Tang Kien	1
1	Developer	1	phuc.dang@axonactive.com	Phuc	male	Dang Hoang	1
2	Information Technology	9	someone.else.2@axonactiv...	Else 2	female	Someone	2
2	Information Technology	8	someone.else.1@axonactiv...	Else 1	female	Someone	2
2	Information Technology	7	kai.vo.1@axonactive.com	Vollmer 1	undefine	Kai	2
2	Information Technology	5	quan.tang.2@axonactive.c...	Quan 2	male	Tang Kien	2
2	Information Technology	3	kai.vo.2@axonactive.com	Vollmer 2	undefine	Kai	2
3	Finance	6	kai.vo.3@axonactive.com	Vollmer 3	male	Kai	3
3	Finance	4	quang.tang.3@axonactive....	Quan 3	female	Tang Kien	3
[null]	[null]	10	someone.else.3@axonactiv...	Else 3	female	Someone	[null]

III. Joins and dynamic fetching:

Full outer join

```
select * from department d full outer join employee e on d.deptId = e.deptid
```

deptid integer	deptname character varying (255)	empid integer	email character varying (255)	firstname character varying (255)	gender character varying (255)	lastname character varying (255)	deptid integer
1	Developer	2	quan.tang.1@axonactive.c...	Quan 1	female	Tang Kien	1
1	Developer	1	phuc.dang@axonactive.com	Phuc	male	Dang Hoang	1
2	Information Technology	9	someone.else.2@axonactiv...	Else 2	female	Someone	2
2	Information Technology	8	someone.else.1@axonactiv...	Else 1	female	Someone	2
2	Information Technology	7	kai.vo.1@axonactive.com	Vollmer 1	undefine	Kai	2
2	Information Technology	5	quan.tang.2@axonactive.c...	Quan 2	male	Tang Kien	2
2	Information Technology	3	kai.vo.2@axonactive.com	Vollmer 2	undefine	Kai	2
3	Finance	6	kai.vo.3@axonactive.com	Vollmer 3	male	Kai	3
3	Finance	4	quang.tang.3@axonactive....	Quan 3	female	Tang Kien	3
4	Human Resources	[null]	[null]	[null]	[null]	[null]	[null]
[null]	[null]	10	someone.else.3@axonactiv...	Else 3	female	Someone	[null]

1. Joins:

inner join

```
session.createCriteria(EmployeeEntity.class)
 .add(Restrictions.eq("gender", "male"))
 .createCriteria("department")
 .add(Restrictions.eq("deptName", "Developer"))
 .list();
```

```
session.createCriteria(EmployeeEntity.class)
 .add(Restrictions.eq("gender", "male"))
 .createAlias("department", "d")
 .add(Restrictions.eq("d.deptName", "Developer")).list();
```

1. Joins:

Full outer join

```
session.createCriteria(DepartmentEntity.class)
        .createAlias("employeeEntities", "e", JoinType.FULL_JOIN)
        .list();
```

ID	Department Name
1	Developer
1	Developer
2	Information Technology
2	Information Technology
2	Information Technology
2	Information Technology
2	Information Technology
3	Finance
3	Finance
4	Human Resources

2. Fetch:

How many types of fetch mode?

```
session.createCriteria(EmployeeEntity.class)
        .setFetchMode("department", FetchMode.SELECT)
        .list();
```

```
select * from employee e
select * from department d where d.deptId = ?
select * from department d where d.deptId = ?
```

```
session.createCriteria(EmployeeEntity.class)
        .setFetchMode("department", FetchMode.JOIN)
        .list();
```

```
select * from employee e left outer join department d on e.deptid = d.deptId
```

IV. Result transformers:

What is result transformers?

- Result transformer: Set a strategy for handling the query results.
  ROOT_ENTITY: default
  DISTINCT_ROOT_ENTITY: distinct
  ALIAS_TO_ENTITY_MAP
  PROJECTION:

IV. Result transformers:

DISTINCT_ROOT_ENTITY

```
session.createCriteria(DepartmentEntity.class)
 .createAlias("employeeEntities", "e", JoinType.FULL_JOIN)
 .setResultTransformer(Criteria.DISTINCT_ROOT_ENTITY).list();
```

ID	Department Name
1	Developer
2	Information Technology
3	Finance
4	Human Resources

IV. Result transformers:

ALIAS_TO_ENTITY_MAP

```
List resultQ = session.createCriteria(DepartmentEntity.class).createAlias("employeeEntities", "e")
        .setFetchMode("e", FetchMode.JOIN).setResultTransformer(Criteria.ALIAS_TO_ENTITY_MAP).list();
List<List<Tuple<String, String>>> result = new ArrayList<>();
for (Object aResult : resultQ) {
    Map map = (Map) aResult;
    DepartmentEntity d = (DepartmentEntity) map.get(Criteria.ROOT_ALIAS);
    EmployeeEntity employeeEntity = (EmployeeEntity) map.get("e");
    result.add(Arrays.asList(
            new Tuple("Department Name ", d.getDeptName()),
            new Tuple("First Name ", employeeEntity.getFirstName() )));
}
```

IV. Projection and report queries:

```
session.createCriteria(EmployeeEntity.class)
 .createAlias("department", "d")
 .setProjection(Projections.projectionList()
         .add(Projections.id().as("id"))
         .add(Projections.property("firstName").as("firstName"))
         .add(Projections.property("lastName").as("lastName"))
         .add(Projections.property("d.deptName").as("departmentName")))
 .setResultTransformer(new AliasToBeanResultTransformer(CustomEmployee.class))
 //.setResultTransformer(Transformers.aliasToBean(CustomEmployee.class))
 .list();
```

```
public class CustomEmployee {
    private Integer id;
    private String firstName;
    private String lastName;
    private String departmentName;
}
```

IV. Projection and report queries:


```
{
    "id": 2,
    "firstName": "Quan 1",
    "lastName": "Tang Kien",
    "departmentName": "Developer"
},
{
    "id": 1,
    "firstName": "Phuc",
    "lastName": "Dang Hoang",
    "departmentName": "Developer"
},
{
    "id": 9,
    "firstName": "Else 2",
    "lastName": "Someone",
    "departmentName": "Information Technology"
},
{
    "id": 8,
    "firstName": "Else 1",
    "lastName": "Someone",
    "departmentName": "Information Technology"
},
```


```
public class CustomEmployee {
    private Integer id;
    private String firstName;
    private String lastName;
    private String departmentName;
}
```

IV. Projection and report queries:

```
session.createCriteria(DepartmentEntity.class)
 .createAlias("employeeEntities", "e")
 .setProjection(Projections.projectionList()
         .add(Property.forName("id").group().as("id"))
         .add(Property.forName("deptName").group().as("name"))
         .add(Property.forName("e.id").avg().as("avgEmployee")))
 .setResultTransformer(new AliasToBeanResultTransformer(CustomDepartment.class))
 .list();
```

```
public class CustomDepartment {
    private Integer id;
    private String name;
    private Double avgEmployee;
}
```

IV. Projection and report queries:


```
{
    "id": 3,
    "name": "Finance",
    "avgEmployee": 5
},
{
    "id": 1,
    "name": "Developer",
    "avgEmployee": 1.5
},
{
    "id": 2,
    "name": "Information Technology",
    "avgEmployee": 6.4
}
```


```
public class CustomDepartment {
    private Integer id;
    private String name;
    private Double avgEmployee;
}
```

That’s it

Actually there are some advanced parts remaining.

But I think you don’t need them.