Java 2

Hibernate

Criteria QueryLanguage

Topics

- Hibernate Criteria Query Language (HCQL)
- Criteria CriteriaBuilder
- Aggregate and Multiple select
- Entity Relationship : Join

HCQL

(Hibernate Criteria Query Language)

HCQL (Hibernate Criteria Query Language)

- Criteria API ถูกออกแบบมาเพื่อให้ผู้ใช้สามารถเขียนคำสั่งเงื่อนไขต่าง ๆ ที่ใช้ในการคิวรีข้อมูลที่ต้องการได้อย่างมีประสิทธิภาพ
- Criteria API ประกอบไปด้วยเมธอดต่าง ๆ ที่ใช้งานบ่อย ๆ ดังต่อไปนี้
  - public Criteria add(Criterion c)
  - public Criteria addOrder(Order o)
  - public Criteria setFirstResult(int firstResult)
  - public Criteria setMaxResult(int totalResult)
  - public List list()
  - public Criteria setProjection(Projection projection)

Criteria API : createQuery()

- Criteria สามารถเรียกใช้อ็อบเจกต์จากคลาส Session ร่วมกับเมธอด createQuery() เพื่อใช้ในการประมวลผลการคิวรีผ่านการเรียกใช้เมธอดต่าง ๆ พร้อมทั้งระบุเงื่อนไขที่ผู้ใช้ต้องการทำงาน

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Student> criteria = builder.createQuery(Student.class);

Root<Student> root = criteria.from(Student.class);

criteria.select(root);
criteria.where( builder.equal( root.get( “name" ), "Somchai Jaidee" ) );

List<Student> results = session.createQuery( criteria ).getResultList();

for (Student stu : results) {
   System.out.println(stu);
}
```

Criteria API : criteria.multiselect();

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<String> criteria = builder.createQuery(String.class);
Root<Student> root = criteria.from(Student.class);

criteria.multiselect(root.get( “name” ));
criteria.where( builder.equal( root.get( “name" ), "Somchai Jaidee" ) );

List<String> results = session.createQuery( criteria ).getResultList();
for (String stu : results) {
    System.out.println(stu);
}
```

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Object[]> criteria = builder.createQuery(Object[].class);
Root<Student> root = criteria.from(Student.class);

Path<Long> idPath = root.get("id");
Path<String> namePath = root.get(“name");

criteria.select( builder.array( idPath, namePath ) );
criteria.where( builder.equal( root.get( “name" ), "Somchai Jaidee" ) );

List<Object[]> results = session.createQuery( criteria ).getResultList();
for (Object[] objs : results) {
    System.out.println("ID: " + objs[0]);
    System.out.println("Name: " + objs[1]);
}
```

Criteria API : Path Class

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Tuple> criteria = builder.createQuery(Tuple.class);
Root<Student> root = criteria.from(Student.class);

Path<Integer> idPath = root.get( "id" );
Path<String> studentNamePath = root.get( “name" );

criteria.multiselect( idPath, studentNamePath );
criteria.where( builder.equal( root.get( “name" ), "Somchai Jaidee" ) );

List<Tuple> results = session.createQuery( criteria ).getResultList();

for (Tuple tuple : results) {
    Long id = tuple.get( idPath );
    String studentName = tuple.get( studentNamePath );
    System.out.println("ID: " + id + ", Name: " + studentName);
}
```

```
…;
for ( Tuple tuple : tuples ) {
    Integer id = (Long) tuple.get( 0 );
    String nickName = (String) tuple.get( 1 );
}
…;
```

Criteria API : Using orderBy

- นอกจากนั้น ยังสามารถใช้ CriteriaBuilder ออปเจ็คเพื่อเรียงลำดับฟิวด์ที่ต้องการได้

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Student> criteria = builder.createQuery(Student.class);
Root<Student> root = criteria.from(Student.class);

criteria.select(root);
criteria.orderBy(builder.asc(root.get("gpa")));

List<Student> results = session.createQuery(criteria).getResultList();

for (Student stu : results) {
   System.out.println(stu);
}
```

```
Student [id=3, name=Somrak Jaijing, age=22, major=Information Technology, gpa=2.5]
Student [id=1, name=Somchai Jaidee, age=22, major=Information Technology, gpa=2.75]
Student [id=4, name=Sompong Jaimon, age=22, major=Computer Science, gpa=3.12]
Student [id=2, name=Somsri Jaipak, age=22, major=Computer Science, gpa=3.25]
```

Criteria CriteriaBuilder

- CriteriaBuilder ยังสามารถใช้ในการจำกัดผลลัพธ์จากคิวรี โดยใช้เมธอดสำคัญ ๆ ดังต่อไปนี้

  กำหนดให้ CriteriaBuilder builder = session.getCriteriaBuilder(); ดังนั้น

  - builder.lt(String propertyName, Object value)
  - builder.le(String propertyName,Object value)
  - builder.gt(String propertyName,Object value)
  - builder.ge(String propertyName,Object value)
  - builder.ne(String propertyName,Object value)
  - builder.equal(String propertyName,Object value)
  - builder.like(String propertyName, Object value)
  - builder.between(String propertyName, Object low, Object high)
  - …

CriteriaBuilder.*equal()*

- เมธอด lt, le, gt, ... คือค่าออกมาเป็นออปเจ็คของคลาส Predicate ซึ่งต้องส่งผ่านเป็นพารามิเตอร์ให้กับเมธอด criteria.where() ของคลาส CriteriaQuery เพื่อเรียกใช้เมธอดต่าง ๆ ที่ใช้ในการจำกัดผลลัพธ์โดยการเปรียบเทียบค่า
- เช่น ในกรณีที่ต้องการอ่านค่าแอททริบิวต์ (property) ที่มีการจำกัดค่าเฉพาะที่เท่ากับ “equals” สามารถทำได้โดยเรียกใช้เมธอด eq() ดังนี้

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Student> criteria = builder.createQuery(Student.class);
Root<Student> root = criteria.from(Student.class);

criteria.select(root);
criteria.where(builder.equal(root.get(“name"), "Somchai Jaidee"));

List<Student> results = session.createQuery(criteria).getResultList();

for (Student stu: results) {
        System.out.println(stu);
}
```

CriteriaBuilder : others

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Student> criteria = builder.createQuery(Student.class);
Root<Student> root = criteria.from(Student.class);

criteria.select(root);
Predicate predicate = builder.equal(root.get("gpa"), 2.5);
criteria.where(predicate);

List<Student> results = session.createQuery(criteria).getResultList();
```

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Student> criteria = builder.createQuery(Student.class);
Root<Student> root = criteria.from(Student.class);

criteria.select(root);
Predicate predicate = builder.greaterThan(root.get("gpa").as(Double.class), 3.0);
criteria.where(predicate);

List<Student> results = session.createQuery(criteria).getResultList();
```

CriteriaBuilder : Using multiple condition

- หากมีเงื่อนไขมากกว่าหนึ่งสามารถทำได้ดังนี้

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Student> criteria = builder.createQuery(Student.class);
Root<Student> root = criteria.from(Student.class);

criteria.select(root);
Predicate [] pds = new Predicate[2];
pds[0] = builder.ge(root.get("gpa").as(Double.class), 2.5);
pds[1] = builder.le(root.get("gpa")  .as(Double.class), 3.0);
criteria.where(pds);

List<Student> results = session.createQuery(criteria).getResultList();
```

```
Student [id=1, name=Somchai Jaidee, age=22, major=Information Technology, gpa=2.75]
Student [id=3, name=Somrak Jaijing, age=22, major=Information Technology, gpa=2.5]
```

CriteriaBuilder.like()

- การค้นหาโดยใช้คำสั่งประเภท SQL LIKE clause ร่วมกับการเรียกใช้เมธอด like() ที่มีลักษณะเป็น case-insensitive ดังตัวอย่างต่อไปนี้

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Student> criteria = builder.createQuery(Student.class);
Root<Student> root = criteria.from(Student.class);

criteria.select(root);
criteria.where(builder.like(root.get("major").as(String.class), "%Inform%"));

List<Student> results = session.createQuery(criteria).getResultList();

for (Student stu: results) {
        System.out.println(stu);
}
```

```
Student [id=1, name=Somchai Jaidee, age=22, major=Information Technology, gpa=2.75]
Student [id=3, name=Somrak Jaijing, age=22, major=Information Technology, gpa=2.5]
```

CriteriaBuilder.between()

- การค้นหาโดยใช้คำสั่งประเภท SQL LIKE clause ร่วมกับการเรียกใช้เมธอด like() ที่มีลักษณะเป็น case-insensitive ดังตัวอย่างต่อไปนี้

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Student> criteria = builder.createQuery(Student.class);
Root<Student> root = criteria.from(Student.class);

criteria.select(root);
Predicate predicate = builder.between(root.get("gpa").as(Double.class), 3.0, 4.0);
criteria.where(predicate);

List<Student> results = session.createQuery(criteria).getResultList();

for (Student stu: results) {
        System.out.println(stu);
}
```

```
Student [id=2, name=Somsri Jaipak, age=22, major=Computer Science, gpa=3.25]
Student [id=4, name=Sompong Jaimon, age=22, major=Computer Science, gpa=3.12]
```

CriteriaBuilder.in()

- การค้นหาโดยใช้คำสั่งประเภท SQL LIKE clause ร่วมกับการเรียกใช้เมธอด like() ที่มีลักษณะเป็น case-insensitive ดังตัวอย่างต่อไปนี้

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Student> criteria = builder.createQuery(Student.class);
Root<Student> root = criteria.from(Student.class);

criteria.select(root);
Predicate predicate = root.get("major").in("Computer Science", "Information Technology");
criteria.where(predicate);

List<Student> results = session.createQuery(criteria).getResultList();

for (Student stu : results) {
   System.out.println(stu);
}
```

```
Student [id=1, name=Somchai Jaidee, age=22, major=Information Technology, gpa=2.75]
Student [id=2, name=Somsri Jaipak, age=22, major=Computer Science, gpa=3.25]
Student [id=3, name=Somrak Jaijing, age=22, major=Information Technology, gpa=2.5]
Student [id=4, name=Sompong Jaimon, age=22, major=Computer Science, gpa=3.12]
```

CriteriaBuilder Custom Function

- หากไม่พบฟังก์ชันที่ต้องการใน CriteriaBuilder เช่น ROUND() ของ MySQL เราสามารถใช้ Custom Function ที่ให้มาได้ เช่น
  - หากต้องการหาค่าเฉลี่ย gpa ของนักศึกษา โดยคำสั่งที่ใช้ใน MySQL คือ ROUND(AVG(gpa), 2) จะเห็นได้ว่ามีการปัดเศษและให้มีทศนิยมสองตำแหน่ง ดังนั้นจะสามารถแปลงให้อยู่ในรูปคำสั่งของ CriteriaBuilder ได้ดังนี้

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Double> criteria = builder.createQuery(Double.class);
Root<Student> root = criteria.from(Student.class);

Expression<Double> avgGPA = builder.function("round",
        Double.class,
        builder.avg(root.get(”gpa")),
        builder.literal(2));

criteria.select(avgGPA);

Double result = session.createQuery(criteria).uniqueResult();
System.out.println(result);
```

SELECT round(gpa,1) FROM it214db.students limit 1;

CriteriaBuilder: Aggregate and Multiple select

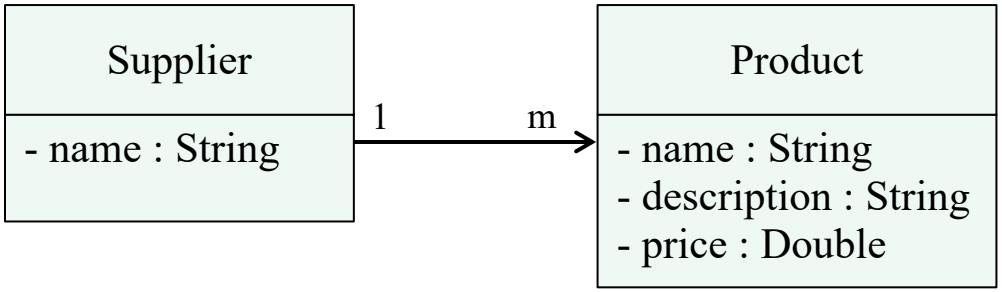
- เราสามารถหาผลลัพธ์จาก Aggregate ฟังก์ชันหลายอันพร้อมกันพร้อมกับการกำหนดเงื่อนไขด้วย CriteraiBuilder

```
CriteriaBuilder builder = session.getCriteriaBuilder();
    CriteriaQuery<Object[]> criteria = builder.createQuery(Object[].class);
    Root<Student> root = criteria.from(Student.class);

    criteria.multiselect(builder.min(root.get("gpa").as(Double.class)),
            builder.max(root.get("gpa").as(Double.class)));
    criteria.where(builder.greaterThan(root.get("age").as(Double.class), 21.0));

    Object[] results = session.createQuery(criteria).getSingleResult();
    System.out.printf("GPA min: %.2f, GPA max: %.2f", results[0], results[1]);
```

GPA min: 3.12, GPA max: 3.25

Suppliers & Product

- กำหนดให้ คลาส Supplier มีความสัมพันธ์แบบหนึ่งต่อกลุ่มไปยังคลาส Product
- ภายในตารางประกอบไปด้วยข้อมูลดังต่อไปนี้



Suppliers & Product

suppliers

id	address	name
1	USA	Procor & Gamble
2	UK	Unilever

products

id	description	name	price	supplier_id
1	Shampoo	Dove	112	1
2	Shampoo	Pantene	120	1
3	Shampoo	Sunsilk	80	1
4	Soap	Dettol	30	1
5	Shampoo	Head & Shoulders	99	1
6	Soap	Lifebuoy	20	1
7	Soap	Lux	25	1
8	fabric softener	Downy	60	2
9	fabric softener	Bounce	55	2
10	laundry detergent	Tide	45	2
11	Soap	Safeguard	35	2
12	laundry detergent	Ariel	50	2
13	fabric freshener	Febreze	70	2

Criteria

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Supplier> criteria = builder.createQuery(Supplier.class);
Root<Supplier> root = criteria.from(Supplier.class);
criteria.select(root);

List<Supplier> results = session.createQuery(criteria).getResultList();

for (Supplier obj : results) {
   System.out.println(obj);
}
```

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Supplier> criteria = builder.createQuery(Supplier.class);
Root<Supplier> root = criteria.from(Supplier.class);

Predicate predicate = builder.like(root.get("name").as(String.class), "%Gamble%");
criteria.select(root);
criteria.where(predicate);

Supplier result = session.createQuery(criteria).uniqueResult();
System.out.println(result);
```

Criteria : Cont.

```java
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Product> criteria = builder.createQuery(Product.class);
Root<Product> root = criteria.from(Product.class);

Predicate predicate = builder.ge(root.get("price").as(Double.class), 100.0);
criteria.select(root);
criteria.where(predicate);

List<Product> results = session.createQuery(criteria).getResultList();
for (Product product : results) {
  System.out.println(product);
}
```

```java
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Long> criteria = builder.createQuery(Long.class);
Root<Product> root = criteria.from(Product.class);

criteria.select(builder.count(root));

Long count = session.createQuery(criteria).uniqueResult();
System.out.println("Row count: " + count);
```

Criteria : Cont.

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Tuple> criteria = builder.createQuery(Tuple.class);
Root<Product> root = criteria.from(Product.class);

Expression<Double> exp1 = builder.max(root.get("price").as(Double.class));
Expression<Double> exp2 = builder.min(root.get("price").as(Double.class));
Expression<Double> exp3 = builder.avg(root.get("price").as(Double.class));
Expression<Long> exp4 = builder.countDistinct(root.get("description"));

criteria.multiselect(exp1, exp2, exp3, exp4);

List<Tuple> results = session.createQuery(criteria).getResultList();
for (Tuple tuple : results) {
  Number max = tuple.get(exp1); // Number is the superclass of Double
  Number min = tuple.get(exp2); // Number is the superclass of Double
  Double avg = tuple.get(exp3);
  Long cdist = tuple.get(exp4);
  System.out.println("Max: " + max + ", Min: " + min + ", Avg: " + avg +
                     ", Count Distinct: " + cdist);
}
```

Max: 120.0, Min: 20.0, Avg: 61.615384615384615, Count Distinct: 5

Criteria : Cont.

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Tuple> criteria = builder.createQuery(Tuple.class);
Root<Product> root = criteria.from(Product.class);

Path<String> productName = root.get("name");
Path<String> productDesc = root.get("description");

criteria.multiselect(productName, productDesc);

List<Tuple> results = session.createQuery(criteria).getResultList();
for (Tuple tuple : results) {
  String pname = tuple.get(productName);
  String pdesc = tuple.get(productDesc);
  System.out.println("" + pname + ", " + pdesc);
}
```

Dove, Shampoo
Pantene, Shampoo

…

Ariel, laundry detergent
Febreze, fabric
freshener

Entity Relationship : Join

Entity Relationship : Join

- **Hibernate** สนับสนุนการทำงานแบบ Join 4 ชนิดดังต่อไปนี้
  - **Inner join** เลือกจับคู่ข้อมูลจากสองอ็อบเจกต์เฉพาะที่มีค่าเท่ากันเท่านั้น inner join เป็นค่า DEFAULT สำหรับการใช้งานร่วมกับ Hibernate
  - **Left join** ใช้สำหรับการเลือกจับคู่ข้อมูลจากทั้งสองอ็อบเจกต์ที่เท่ากันของการ join โดยเน้นไปที่อ็อบเจกต์ที่อยู่ทางด้านซ้ายมือเป็นหลัก แม้ว่าไม่มีอ็อบเจกต์ที่เท่ากันปรากฏอยู่ทางด้านขวามือก็ตาม
  - **Right join** ใช้สำหรับการเลือกจับคู่ข้อมูลจากทั้งสองอ็อบเจกต์ที่เท่ากันของการ join โดยเน้นไปที่อ็อบเจกต์ที่อยู่ทางด้านขวามือเป็นหลัก แม้ว่าไม่มีอ็อบเจกต์ที่เท่ากันปรากฏอยู่ทางด้านซ้ายมือก็ตาม
  - **Full join** ใช้สำหรับการเลือกจับคู่ข้อมูลจากทั้งสองอ็อบเจกต์ทั้งที่เท่ากันและไม่เท่ากันจากทั้งสองด้าน

Bidirectional 1 : M

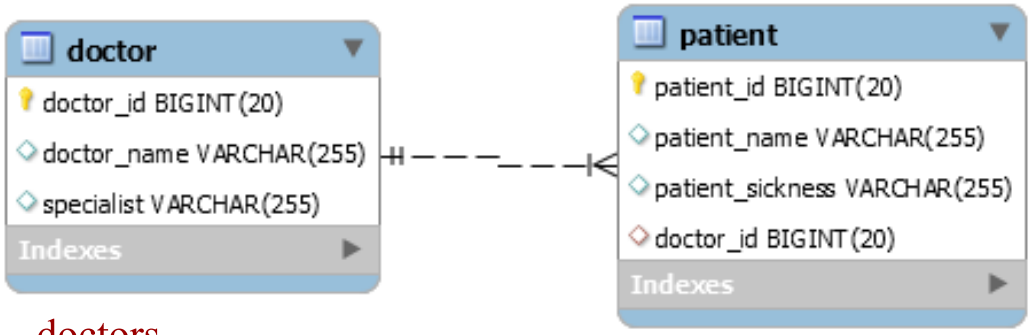
```
@Entity
@Table(name = "doctors")
public class Doctor {
    ...;

    @OneToMany (cascade=CascadeType.ALL, mappedBy = "doctor")
    private Set<Patient> patients = new HashSet<Patient>();
}
```

```
@Entity
@Table (name = ”patients")
public class Patient {
    @ManyToOne
    @JoinColumn (name = "doctor_id")
    private Doctor doctor;

    …;
}
```

Data in DB

```
Doctor doctor1 = new Doctor("Dr. Sombat Jaidee", "General Practitioner");
Doctor doctor2 = new Doctor("Dr. Somchai Jaipak", "Allergist");
Doctor doctor3 = new Doctor("Dr. Somrak Jaijing", "Surgery");
Patient patient1 = new Patient("Somsri Jaipak", "Stomachache");
Patient patient2 = new Patient("Sompong Jaimon", "Headache");
Patient patient3 = new Patient("Somchai Jaidee", "Heart attack");
Patient patient4 = new Patient("Somrak Jaijing", "Broken leg");

doctor1.getPatients().add(patient1);
doctor1.getPatients().add(patient2);
patient1.setDoctor(doctor1);
patient2.setDoctor(doctor1);

doctor2.getPatients().add(patient3);
doctor2.getPatients().add(patient4);
patient3.setDoctor(doctor2);
patient4.setDoctor(doctor2);

session.save(doctor1);
session.save(doctor2);
session.save(doctor3);
```

Data in DB



doctors

id	name	specialist
1	Dr. Sombat Jaidee	General Practitioner
2	Dr. Somchai Jaipak	Allergist
3	Dr. Somrak Jaijing	Surgery

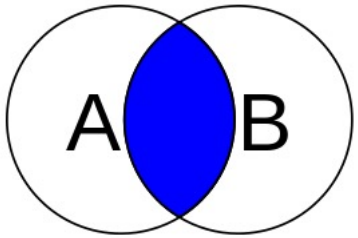
patients

id	name	sickness	doctor_id
1	Sompong Jaimon	Headache	1
2	Somsri Jaipak	Stomachache	1
3	Somrak Jaijing	Broken leg	2
4	Somchai Jaidee	Heart attack	2

Inner Join

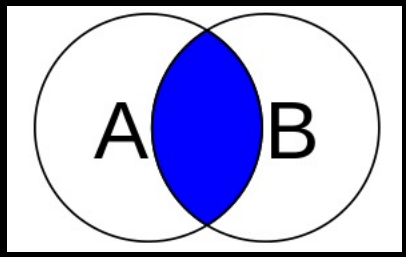
```
String hql = "from Doctor as D inner join D.patients as P";
Query<Object[]> query = session.createQuery(hql, Object[].class);

List<Object[]> list = query.getResultList();

for (int i = 0; i < list.size(); i++) {
  Object[] row = list.get(i);
  Doctor doctor = (Doctor) row[0];
  Patient patient = (Patient) row[1];
  System.out.println(doctor + " => " + patient.getName());
}
```

```
Doctor [id=1, name=Dr. Sombat Jaidee, specialist=General Practitioner] => Sompong Jaimon
Doctor [id=1, name=Dr. Sombat Jaidee, specialist=General Practitioner] => Somsri Jaipak
Doctor [id=2, name=Dr. Somchai Jaipak, specialist=Allergist] => Somrak Jaijing
Doctor [id=2, name=Dr. Somchai Jaipak, specialist=Allergist] => Somchai Jaidee
```



Inner Join Using CriteriaBuilder

```
CriteriaBuilder cb = session.getCriteriaBuilder();
CriteriaQuery<Tuple> cq = cb.createTupleQuery();
Root<Doctor> root = cq.from(Doctor.class);

Join<Object, Object> patientJoin = root.join("patients", JoinType.INNER);
cq.multiselect(root, patientJoin);
TypedQuery<Tuple> q = session.createQuery(cq);

List<Tuple> tuples = q.getResultList();

for (Tuple tuple : tuples) {
  Doctor doctor = (Doctor) tuple.get(0);
  Patient patient = (Patient) tuple.get(1);
  System.out.print(doctor);
  if (patient != null)
    System.out.println(" ==> " + patient);
}
```




```
Doctor [id=1, name=Dr. Sombat Jaidee, specialist=General Practitioner] ==> Patient [id=1, name=Sompong Jaimon, sickness=Headache]
Doctor [id=1, name=Dr. Sombat Jaidee, specialist=General Practitioner] ==> Patient [id=2, name=Somsri Jaipak, sickness=Stomachache]
Doctor [id=2, name=Dr. Somchai Jaipak, specialist=Allergist] ==> Patient [id=3, name=Somrak Jaijing, sickness=Broken leg]
Doctor [id=2, name=Dr. Somchai Jaipak, specialist=Allergist] ==> Patient [id=4, name=Somchai Jaidee, sickness=Heart attack]
```

Inner Join

```
String hql = "SELECT d, p FROM Doctor d JOIN d.patients p WHERE p.sickness
LIKE '%ache%'";

Query<Object[]> query = session.createQuery(hql, Object[].class);
List<Object[]> results = query.getResultList();

for (Object[] objs : results) {
  Doctor doctor = (Doctor) objs[0];
  Patient patient = (Patient) objs[1];
  System.out.println(doctor);
  System.out.println("**** " + patient);
}
```

```
Doctor [id=1, name=Dr. Sombat Jaidee, specialist=General Practitioner]
**** Patient [id=1, name=Sompong Jaimon, sickness=Headache]
Doctor [id=1, name=Dr. Sombat Jaidee, specialist=General Practitioner]
**** Patient [id=2, name=Somsri Jaipak, sickness=Stomachache]
```

Inner Join Using CriteriaBuilder

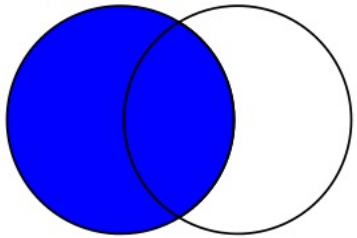
```
CriteriaBuilder cb = session.getCriteriaBuilder();
CriteriaQuery<Tuple> cq = cb.createTupleQuery();
Root<Doctor> root = cq.from(Doctor.class);
Join<Object, Object> patientJoin = root.join("patients", JoinType.INNER);

cq.multiselect(root, patientJoin);

ParameterExpression<String> sickness = cb.parameter(String.class);
cq.where(cb.like(patientJoin.get("sickness").as(String.class), sickness));

TypedQuery<Tuple> q = session.createQuery(cq);
q.setParameter(sickness, "%ache%");
List<Tuple> tuples = q.getResultList();

for (Tuple tuple : tuples) {
  Doctor doctor = (Doctor) tuple.get(0);
  Patient patient = (Patient) tuple.get(1);
  System.out.println(doctor);
  System.out.println("**** " + patient);
}
```

Doctor Id: 1, doctor Name: Dr. Sombat Jaidee
**** Patient Id: 1, Patient Name: Sompong Jaimon, Sickness: Headache
Doctor Id: 1, doctor Name: Dr. Sombat Jaidee
**** Patient Id: 2, Patient Name: Somsri Jaipak, Sickness: Stomachache

Left OUTER JOIN

```
String hql = "select D, P from Doctor as D left outer join D.patients as P";
Query<Object[]> query = session.createQuery(hql, Object[].class);
List<Object[]> list = query.getResultList();

for (Object[] row : list) {
  Doctor doctor = (Doctor) row[0];
  Patient patient = (Patient) row[1];
  System.out.println(doctor);
  if (patient != null)
    System.out.println(" ==> " + patient);
}
```

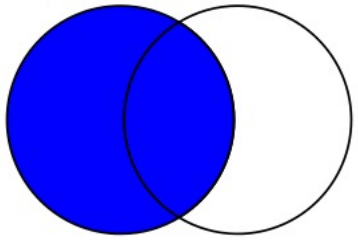
```
Doctor [id=1, name=Dr. Sombat Jaidee, specialist=General Practitioner]
 ==> Patient [id=2, name=Somsri Jaipak, sickness=Stomachache]
Doctor [id=1, name=Dr. Sombat Jaidee, specialist=General Practitioner]
 ==> Patient [id=1, name=Sompong Jaimon, sickness=Headache]
Doctor [id=2, name=Dr. Somchai Jaipak, specialist=Allergist]
 ==> Patient [id=4, name=Somchai Jaidee, sickness=Heart attack]
Doctor [id=2, name=Dr. Somchai Jaipak, specialist=Allergist]
 ==> Patient [id=3, name=Somrak Jaijing, sickness=Broken leg]
Doctor [id=3, name=Dr. Somrak Jaijing, specialist=Surgery]
```



Left OUTER JOIN Using CriteriaBuilder

```
CriteriaBuilder cb = session.getCriteriaBuilder();
CriteriaQuery<Tuple> cq = cb.createTupleQuery();
Root<Doctor> root = cq.from(Doctor.class);
Join<Object, Object> patientJoin = root.join("patients", JoinType.LEFT);

cq.multiselect(root, patientJoin);

TypedQuery<Tuple> q = session.createQuery(cq);
List<Tuple> tuples = q.getResultList();

for (Tuple tuple : tuples) {
  Doctor doctor = (Doctor) tuple.get(0);
  Patient patient = (Patient) tuple.get(1);
  System.out.println(doctor);
  if (patient != null)
 System.out.println(" ==> " + patient);
}
```

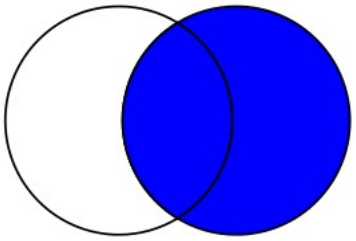
```
Doctor [id=1, name=Dr. Sombat Jaidee, specialist=General Practitioner]
 ==> Patient [id=2, name=Somsri Jaipak, sickness=Stomachache]
Doctor [id=1, name=Dr. Sombat Jaidee, specialist=General Practitioner]
 ==> Patient [id=1, name=Sompong Jaimon, sickness=Headache]
Doctor [id=2, name=Dr. Somchai Jaipak, specialist=Allergist]
 ==> Patient [id=4, name=Somchai Jaidee, sickness=Heart attack]
Doctor [id=2, name=Dr. Somchai Jaipak, specialist=Allergist]
 ==> Patient [id=3, name=Somrak Jaijing, sickness=Broken leg]
Doctor [id=3, name=Dr. Somrak Jaijing, specialist=Surgery]
```



Right OUTER JOIN

```
String hql = "select D, P from Doctor as D right outer join D.patients as
P";
Query<Object[]> query = session.createQuery(hql, Object[].class);
List<Object[]> list = query.getResultList();

for (Object[] row : list) {
  Doctor doctor = (Doctor) row[0];
  Patient patient = (Patient) row[1];
  System.out.println(doctor);
  if (patient != null)
    System.out.println(" ==> " + patient);
}
```

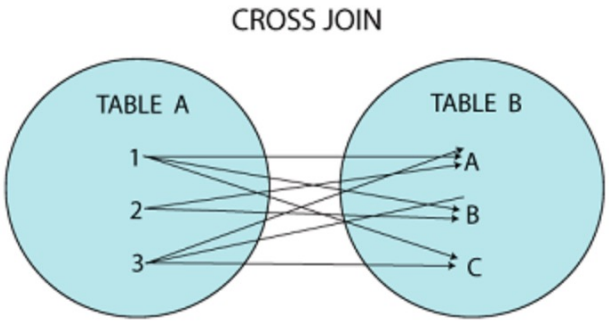
```
Doctor [id=1, name=Dr. Sombat Jaidee, specialist=General Practitioner]
 ==> Patient [id=1, name=Sompong Jaimon, sickness=Headache]
Doctor [id=1, name=Dr. Sombat Jaidee, specialist=General Practitioner]
 ==> Patient [id=2, name=Somsri Jaipak, sickness=Stomachache]
Doctor [id=2, name=Dr. Somchai Jaipak, specialist=Allergist]
 ==> Patient [id=3, name=Somrak Jaijing, sickness=Broken leg]
Doctor [id=2, name=Dr. Somchai Jaipak, specialist=Allergist]
 ==> Patient [id=4, name=Somchai Jaidee, sickness=Heart attack]
```



Cross JOIN

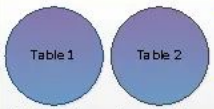
```
String hql = "from Doctor as D, Patient as P";
Query<Object[]> query = session.createQuery(hql, Object[].class);
List<Object[]> list = query.getResultList();

for (Object[] row : list) {
  Doctor doctor = (Doctor) row[0];
  Patient patient = (Patient) row[1];
  System.out.print(doctor.getName());
  if (patient != null)
    System.out.println(" ==> " + patient.getName());
}
```

Dr. Somrak Jaijing ==> Sompong Jaimon
Dr. Somchai Jaipak ==> Sompong Jaimon
Dr. Sombat Jaidee ==> Sompong Jaimon
Dr. Somrak Jaijing ==> Somsri Jaipak
Dr. Somchai Jaipak ==> Somsri Jaipak
Dr. Sombat Jaidee ==> Somsri Jaipak
Dr. Somrak Jaijing ==> Somrak Jaijing
Dr. Somchai Jaipak ==> Somrak Jaijing
Dr. Sombat Jaidee ==> Somrak Jaijing
Dr. Somrak Jaijing ==> Somchai Jaidee
Dr. Somchai Jaipak ==> Somchai Jaidee
Dr. Sombat Jaidee ==> Somchai Jaidee



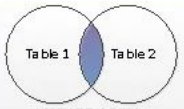
MySQL JOIN Types

Created by Steve Stedman



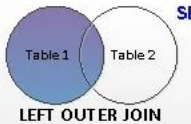
SELECT from two tables

```
SELECT *
  FROM Table1;

SELECT *
  FROM Table2;
```



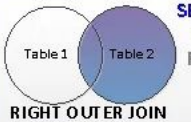
INNER JOIN

```
SELECT *
  FROM Table1 t1
 INNER JOIN Table2 t2
    ON t1.fk = t2.id;
```



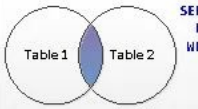
LEFT OUTER JOIN

```
SELECT *
  FROM Table1 t1
 LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id;
```



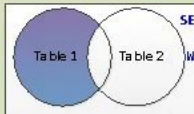
RIGHT OUTER JOIN

```
SELECT *
  FROM Table1 t1
 RIGHT OUTER JOIN Table2 t2
    ON t1.fk = t2.id;
```



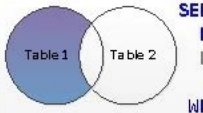
SEMI JOIN – Similar to INNER JOIN, with less duplication.

```
SELECT *
  FROM Table1 t1
 WHERE EXISTS (SELECT 1
                 FROM Table2 t2
                WHERE t1.fk = t2.id
               );
```



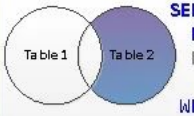
ANTI SEMI JOIN

```
SELECT *
  FROM Table1 t1
 WHERE NOT EXISTS (SELECT 1
                     FROM Table2 t2
                    WHERE t1.fk = t2.id
                   );
```



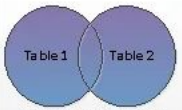
LEFT OUTER JOIN with exclusion

```
SELECT *
  FROM Table1 t1
 LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 WHERE t2.id is null;
```



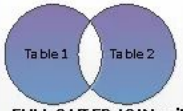
RIGHT OUTER JOIN with exclusion

```
SELECT *
  FROM Table1 t1
 RIGHT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 WHERE t1.fk is null;
```



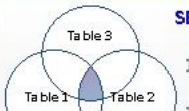
FULL OUTER JOIN

```
SELECT * FROM Table1 t1
  LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
UNION
SELECT * FROM Table1 t1
 RIGHT OUTER JOIN Table2 t2
    ON t1.fk = t2.id;
```



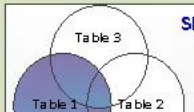
FULL OUTER JOIN with exclusion

```
SELECT * FROM Table1 t1
  LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 WHERE t2.id IS NOT NULL
UNION
SELECT * FROM Table1 t1
 RIGHT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 WHERE t1.ID IS NOT NULL;
```



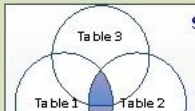
Two INNER JOINs

```
SELECT *
  FROM Table1 t1
 INNER JOIN Table2 t2
    ON t1.fk = t2.id
 INNER JOIN Table3 t3
    ON t1.fk_table3 = t3.id;
```



Two LEFT OUTER JOINS

```
SELECT *
  FROM Table1 t1
 LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 LEFT OUTER JOIN Table3 t3
    ON t1.fk_table3 = t3.id;
```



INNER JOIN and a LEFT OUTER JOIN

```
SELECT *
  FROM Table1 t1
 INNER JOIN Table2 t2
    ON t1.fk = t2.id
 LEFT OUTER JOIN Table3 t3
    ON t1.fk_table3 = t3.id;
```

Good luck

Refference:

https://hibernate.org/orm/releases/5.6/

https://hibernate.org/orm/documentation/5.6/

https://docs.jboss.org/hibernate/stable/orm/userguide/html_single/Hibernate_User_Guide.html

https://github.com/brmeyer/HibernateDemos