# 单向多对一

## 概念

只有一方维护

## jpa配置

一个订单只有一个用户，一个用户多个订单

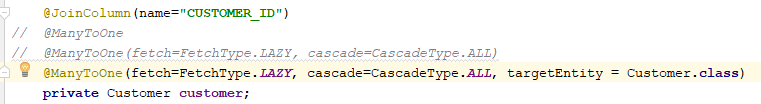
使用 @ManyToOne 来映射多对一的关联关系， fetch 属性来修改默认的关联属性的加载策略

使用 @JoinColumn 来映射外键.

@JoinColumn(name="CUSTOMER\_ID")

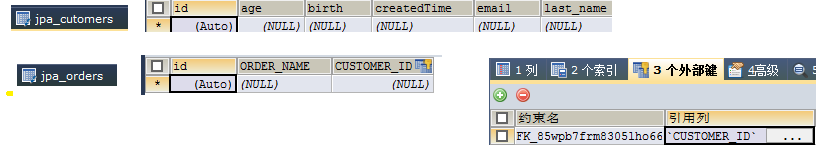
@ManyToOne(fetch=FetchType.***LAZY***)





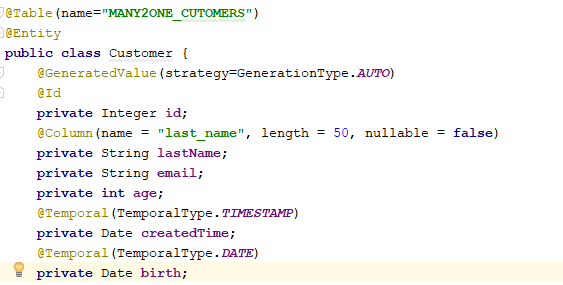
## 数据生成信息

Order表会产生外键



## 测试准备

### Customer



### Order



### 准备JPATest

**public** **class** JPATest {

**private** EntityManagerFactory entityManagerFactory;

**private** EntityManager entityManager;

**private** EntityTransaction transaction;

@Before

**public** **void** init(){

entityManagerFactory = Persistence.*createEntityManagerFactory*("jpa\_test1");

entityManager = entityManagerFactory.createEntityManager();

transaction = entityManager.getTransaction();

transaction.begin();

}

@After

**public** **void** destroy(){

transaction.commit();

entityManager.close();

entityManagerFactory.close();

}

}

# Jpa操作

## 保存

建议先保存一的一方，再保存多的一方，就不会出现update语句

### 测试先多再一

先保存一的一端，再保存多的一端

/\*\*

\* 保存多对一时, 建议先保存 1 的一端, 后保存 n 的一端, 这样不会多出额外的 UPDATE 语句.

\*/

@Test

**public** **void** testManyToOnePersist(){

Customer customer = **new** Customer();

customer.setAge(18);

customer.setBirth(**new** Date());

customer.setCreatedTime(**new** Date());

customer.setEmail("gg@163.com");

customer.setLastName("GG");

Order order1 = **new** Order();

order1.setOrderName("G-GG-1");

Order order2 = **new** Order();

order2.setOrderName("G-GG-2");

//设置关联关系

order1.setCustomer(customer);

order2.setCustomer(customer);

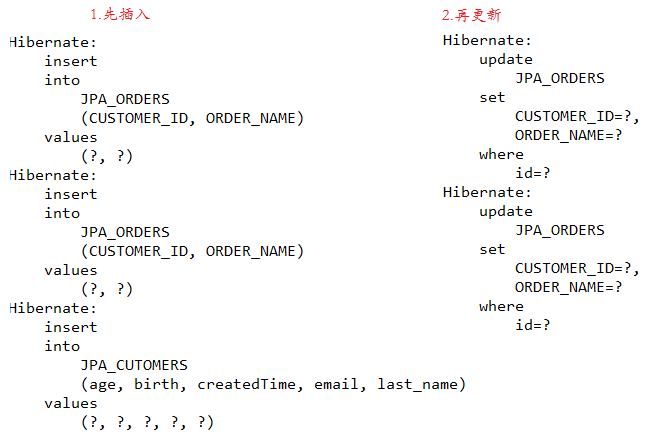
//执行保存操作

entityManager.persist(order1);

entityManager.persist(order2);

entityManager.persist(customer);

}



### 测试先一再多

/\*\*

\* 保存多对一时, 建议先保存 1 的一端, 后保存 n 的一端, 这样不会多出额外的 UPDATE 语句.

\*/

@Test

**public** **void** testManyToOnePersist2(){

Customer customer = **new** Customer();

customer.setAge(18);

customer.setBirth(**new** Date());

customer.setCreatedTime(**new** Date());

customer.setEmail("gg@163.com");

customer.setLastName("GG");

Order order1 = **new** Order();

order1.setOrderName("G-GG-1");

Order order2 = **new** Order();

order2.setOrderName("G-GG-2");

//设置关联关系

order1.setCustomer(customer);

order2.setCustomer(customer);

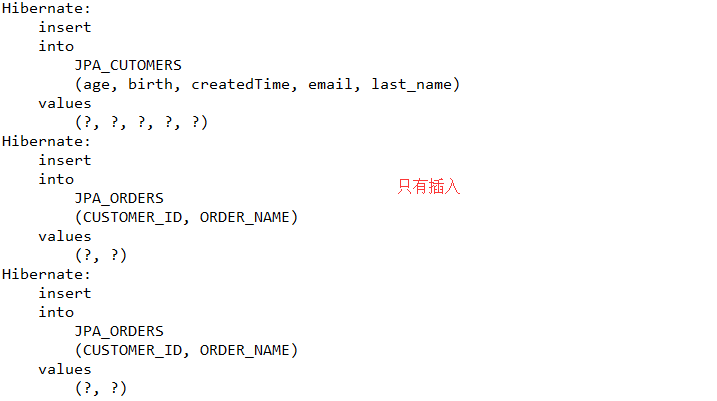
//执行保存操作(先一再多)

entityManager.persist(customer);

entityManager.persist(order1);

entityManager.persist(order2);

}



## 查询（oid）

### 默认加载

查询多方

//默认情况下, 使用左外连接的方式来获取 n 的一端的对象和其关联的 1 的一端的对象.

//可使用 @ManyToOne 的 fetch 属性来修改默认的关联属性的加载策略

@Test

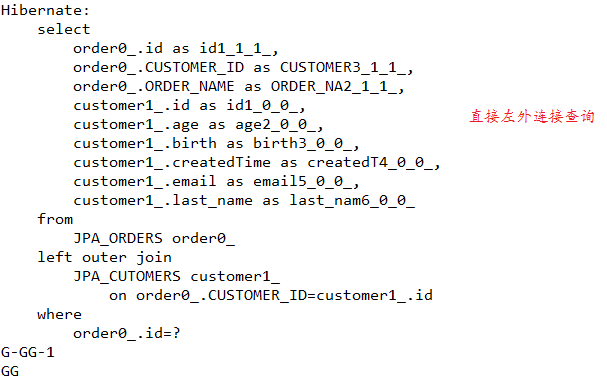
**public** **void** testManyToOneFind(){

Order order = entityManager.find(Order.**class**, 1);

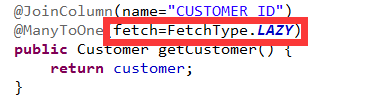
System.***out***.println(order.getOrderName());

System.***out***.println(order.getCustomer().getLastName());

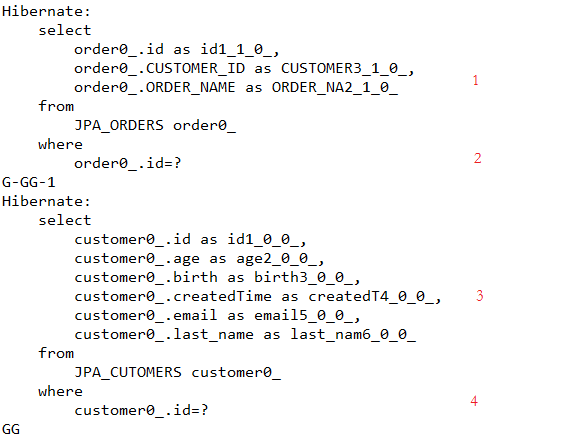
}



### 懒加载



懒加载：当调用时，才发送查询sql语句



## 删除（多方）

只能删除多的一方

//不能直接删除 1 的一端, 因为有外键约束.

@Test

**public** **void** testManyToOneRemove(){

Order order = entityManager.find(Order.**class**, 1);

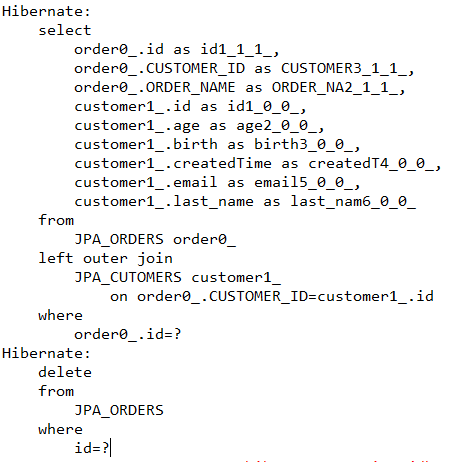
entityManager.remove(order);

// 不能删除1的一方

// Customer customer = entityManager.find(Customer.class, 1);

// entityManager.remove(customer);

}



## 更新

只要是针对多方，一方就直接单表了

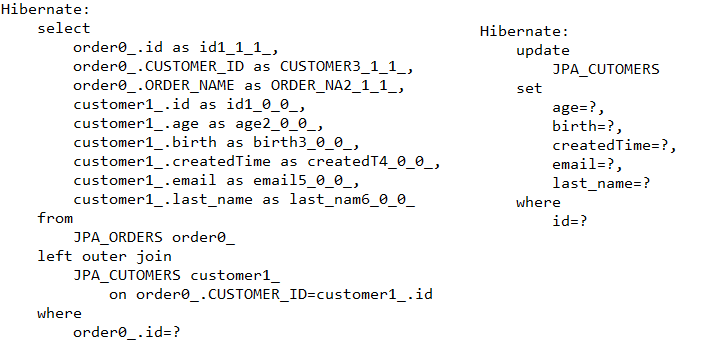
@Test

**public** **void** testManyToOneUpdate(){

Order order = entityManager.find(Order.**class**, 2);

order.getCustomer().setLastName("FFF");

}



# Spring Data Jpa操作

## 原理

由于只有多方配置了关系，所以级联只能多方维护

## 级联

### 级联保存（只能先一再多）

先多再一： 保存多对一时, 建议先保存 1 的一端, 后保存 n 的一端, 这样不会多出额外的 UPDATE 语句.  
先保存多，再一，对于自增主键，会报错

### 级联删除（删多）

删一会报错

### 查询/级联查询

涉及懒加载问题，以及事务管理