# 表关联查询差异

表关联查询主要有两种方式：  
（1）使用外键约束；  
（2）应用层实现关联查询逻辑；

## 外键约束的优势与劣势

### 优点：

1. 保证数据的完整性和一致性
2. 级联操作方便
3. 将数据完整性判断托付给了数据库完成，减少了程序的代码量

### 缺点：

1. 性能问题：每次修改表时，都会查询关联表是否有相应信息，不必要的查询过程会影响系统性能
2. 并发问题：修改数据时会获取关联表的锁，高并发情况下更容易造成死锁
3. 扩展性问题：（1）数据库迁移时，如MySql->Oracle，外键关系的迁移不方便；（2）分库分表时，外键无法生效，分表不方便
4. 技术问题：外键将判断逻辑由程序转移到数据库上，需要专业的数据库维护人员才能优化性能

### 比较：

使用外键可以简化代码、保证数据一致性和完整性，但是不灵活且有可能造成性能下降。  
应用层实现关联查询逻辑程序设计灵活，性能优化方法较多，但需要用代码实现逻辑。  
程序设计初期，可以使用外键增加开发效率。后期程序体量增加之后，外键会成为系统性能瓶颈，必须改造成应用层实现关联查询。  
总的来说，要尽量避免使用外键

## Jhipster & JPA

jhipster使用的是JPA注解实现关联查询，**是使用外键的**，是双向关联关系  
JPA中，数据库中，表之间的关联是通过外键做到的。实体中，类之间的关联是通过注解来实现的。

* 需要外键  
  jhipster使用较多的是joincomlum+mapperBy的注解来完成表关联关系，这种方法是使用外键查询的，其中ManyToMany关系中，会由JPA建立一张中间表来完成多对多关联。
* 应用层实现表关联  
  jhipster没有提供应用层逻辑实现表关联查询功能，需要自己实现
* JPA提供的双向关联使得双方都持有对方的实例，使用较方便
* 使用外键时，只需要按照JPA规范提供接口方法名，JPA会自动实现关联查询方法

## Jeecg-Boot & mybatis-plus

* 代码生成器**用到了外键的概念**  
  利用外键的概念设计出数据表之间的关系
* **没有实际给数据库加外键约束**  
  在实际生成数据库表和代码时使用的是应用层逻辑关联数据表，没有使用到外键。

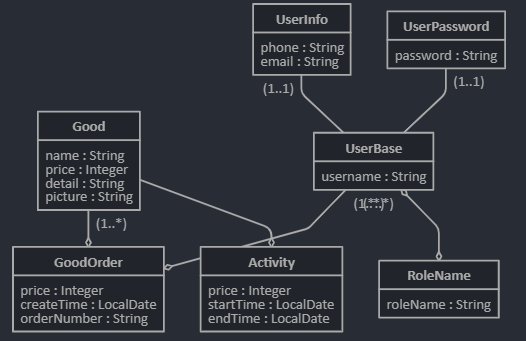
## 总结

|  |  |  |
| --- | --- | --- |
| 表关联方式 | JPA | MyBatis-Plus |
| 外键约束 | 支持 | 好像不支持 |
| 应用层逻辑 | 支持 | 支持 |
| sql语句关联 | 支持 | 支持 |

|  |  |  |
| --- | --- | --- |
| 对比 | Jhipster | Jeecg-Boot |
| 数据库外键约束 | 有 | 无 |
| 关联方式 | 接口方法名查询 | sql语句查询 |

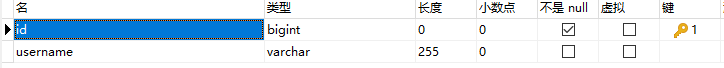
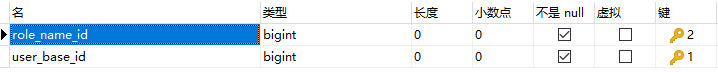
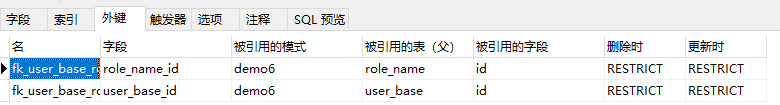
Jhipster的表关联默认生成的是带有外键约束的数据表，然后使用JPA的规范写出接口方法来实现表关联；Jeecg-Boot的表关联默认生成的是不带外键约束的数据表，然后编写sql语句实现关联查询。  
外键约束保证数据的完整性和一致性，但同时也带来了耦合度高的问题，对程序的扩展不利。因此Jeecg-Boot相比Jhipster采用的表关联代码可能更优势。

## Jhipster + JPA建立关联表实例

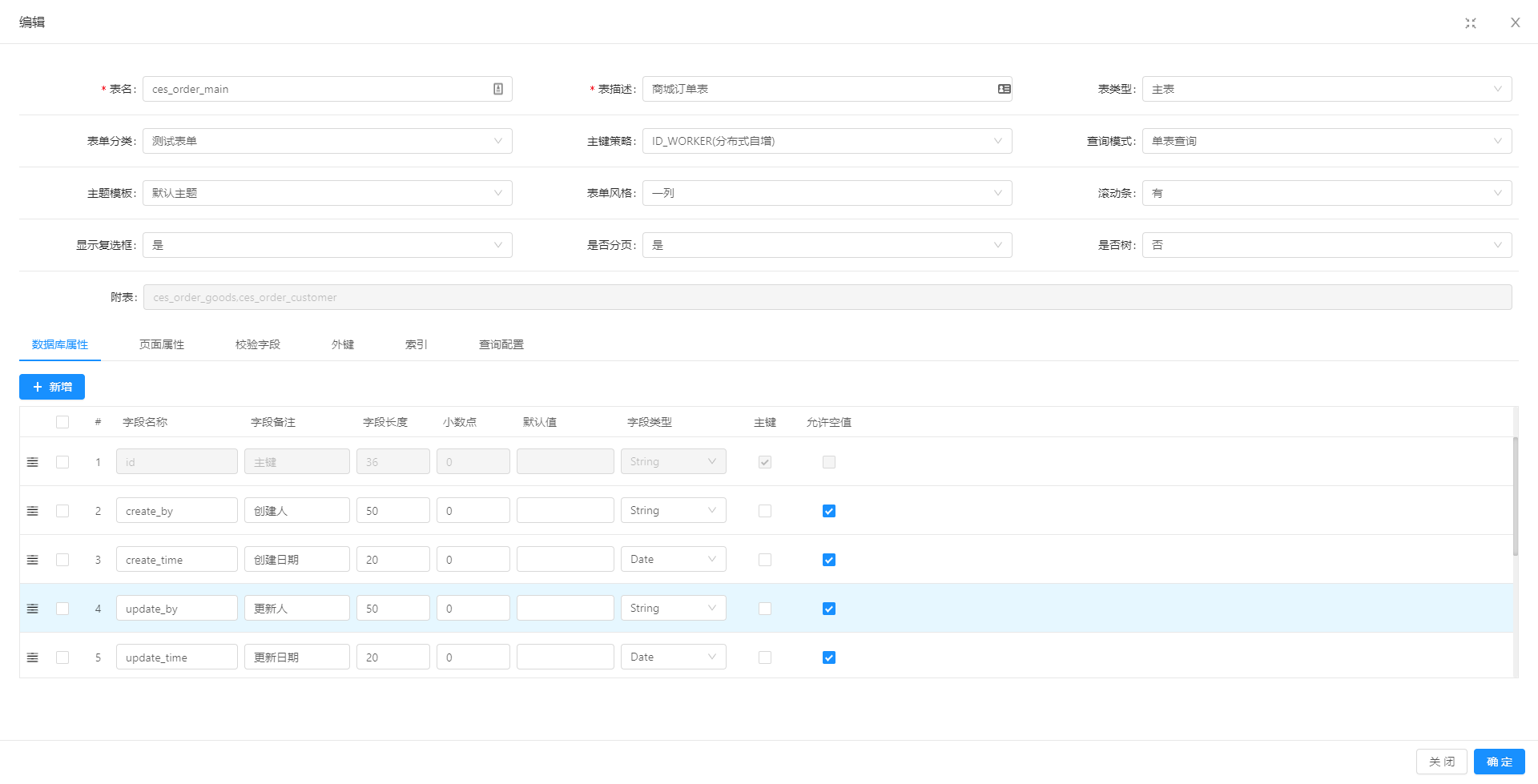
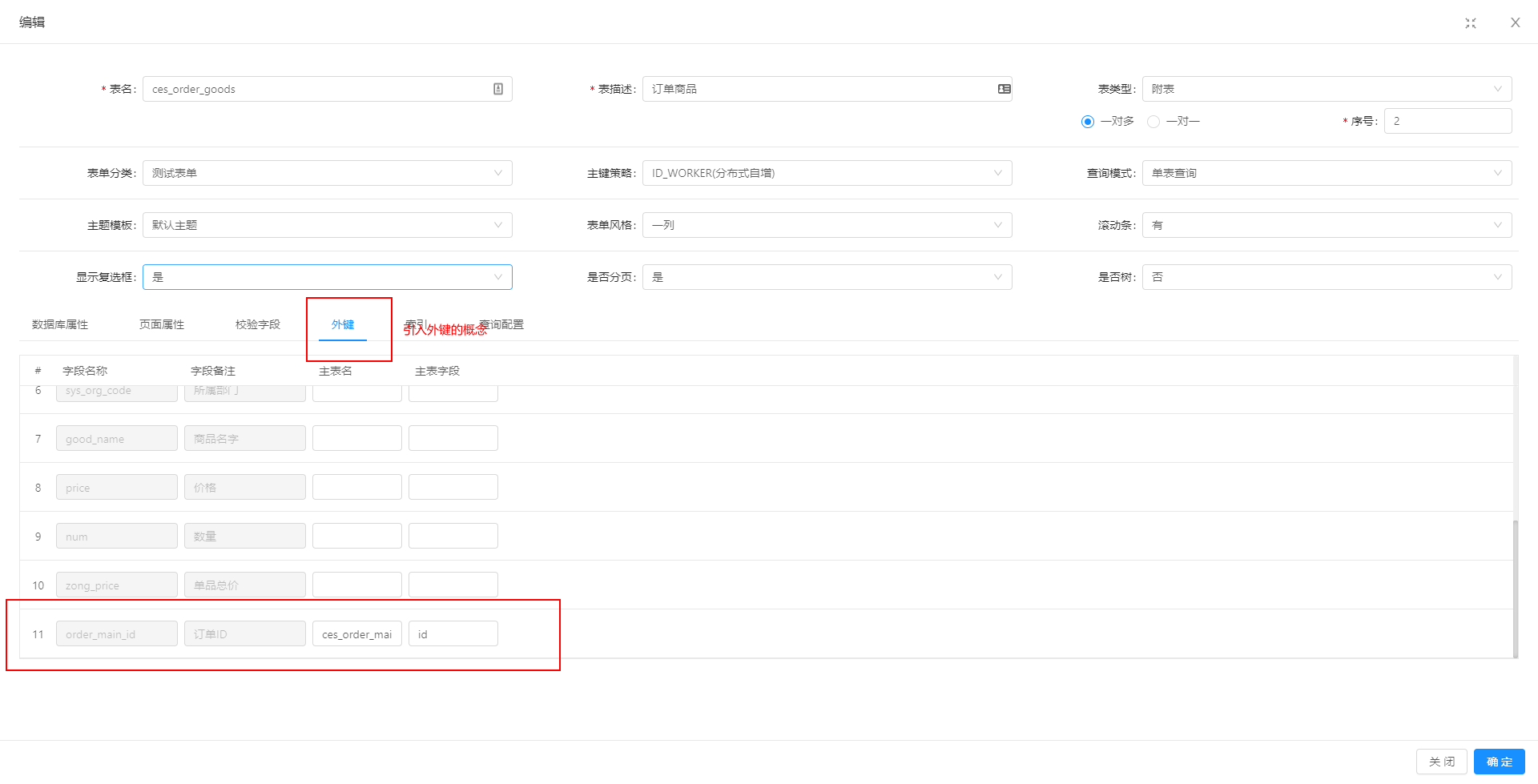
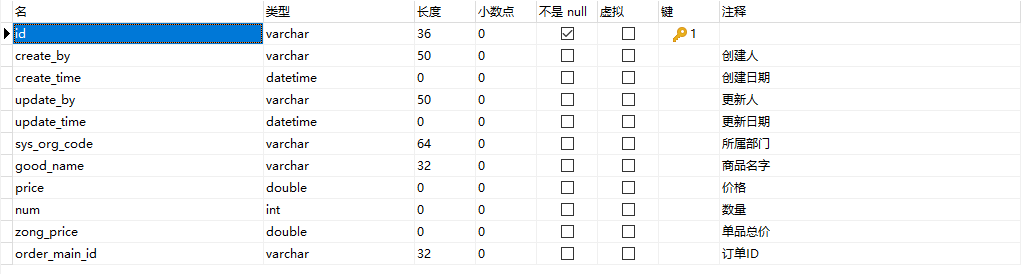


image

// 商品  
entity Good {  
 name String,  
 price Integer,  
 detail String,  
 picture String,  
}  
  
// 订单  
entity GoodOrder {  
 price Integer,  
 createTime LocalDate,  
 orderNumber String  
}  
// 促销活动  
entity Activity {  
 price Integer,  
 startTime LocalDate,  
 endTime LocalDate,  
}  
// 用户详细信息  
entity UserInfo {  
 phone String minlength(11) maxlength(11),  
 email String,  
}  
// 用户密码表  
entity UserPassword {  
 password String,  
}  
  
// 中间表  
//entity UserRole {  
//}  
  
// 角色权限表  
entity RoleName {  
 roleName String  
}  
  
// 用户基本信息表  
entity UserBase {  
 username String  
}  
  
// OneToMany 外键在GoodOrder  
relationship OneToMany {  
 Good to GoodOrder{good(id)}  
 Good to Activity{good(id)}  
 UserBase to GoodOrder{userBase(id)}  
}  
  
// OneToOne 外键在UserInfo  
relationship OneToOne {  
 UserInfo to UserBase,  
 UserPassword to UserBase,  
}  
  
// ManyToMany 外键在UserBase  
relationship ManyToMany {  
 UserBase to RoleName  
}

**user\_base** 该表是用户基本信息表，主表  **user\_info** 该表外键为user\_base\_id  **role\_name**  **user\_base\_role\_name** 该表是user\_base和role\_name多对多关系的中间表   
外键约束 

## Jeecg-Boot + MyBatis-Plus建立关联表实例

如下面例子，订单是主表，商品和顾客是附表  
**生成订单实体类页面**   
**生成商品实体类页面**   
**order表**   
**good表** 

// 一个由Jeecg-Boot生成的保存订单的service方法。订单、商品、顾客  
 @Override  
 @Transactional  
 public void saveMain(CesOrderMain cesOrderMain, List<CesOrderGoods> cesOrderGoodsList,List<CesOrderCustomer> cesOrderCustomerList) {  
 cesOrderMainMapper.insert(cesOrderMain);  
 if(cesOrderGoodsList!=null && cesOrderGoodsList.size()>0) {  
 for(CesOrderGoods entity:cesOrderGoodsList) {  
 //外键设置  
 entity.setOrderMainId(cesOrderMain.getId());  
 cesOrderGoodsMapper.insert(entity);  
 }  
 }  
 if(cesOrderCustomerList!=null && cesOrderCustomerList.size()>0) {  
 for(CesOrderCustomer entity:cesOrderCustomerList) {  
 //外键设置  
 entity.setOrderMainId(cesOrderMain.getId());  
 cesOrderCustomerMapper.insert(entity);  
 }  
 }  
 }