操作记录

1.关联说明

1.一对一（单向）

1.关键实体

```
People -> t_people

idCard -> t_idcard 维护外键
@OneToOne(cascade = {CascadeType.PERSIST})
@JoinColumn(name = "t_people_id")//外键id
private People people;
```

2.自动生成的表结构

```
CREATE TABLE `t_person` (
  `t_id` int(11) NOT NULL,
  `t_address` varchar(255) DEFAULT NULL,
  `t_age` int(11) DEFAULT NULL,
  `t_birthday` datetime(6) DEFAULT NULL,
  `t_name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`t_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;


CREATE TABLE `t_idcard` (
  `t_id` int(11) NOT NULL AUTO_INCREMENT,
  `t_number` varchar(255) DEFAULT NULL,
  `t_people_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`t_id`),
  KEY `FK7gdvrysil6gxmt806ysqr8atn` (`t_people_id`),
  CONSTRAINT `FK7gdvrysil6gxmt806ysqr8atn` FOREIGN KEY (`t_people_id`)
REFERENCES `t_people` (`t_id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;
```

2.一对一（双向）

1.关键实体

```
People2 -> t_people2
@OneToOne(mappedBy = "people2")//负责维护外键的表对应的实体中，持有的本类类型属性的属性名字
    private IdCard2 idCard2;


idCard2 -> t_idcard2 维护外键

    @OneToOne(cascade = {CascadeType.PERSIST})
    @JoinColumn(name = "t_people_id")//外键id
    private People2 people2;
```

2.自动生成的表结构

```
CREATE TABLE `t_people2` (
  `t_id` int(11) NOT NULL AUTO_INCREMENT,
  `t_age` int(11) DEFAULT NULL,
  `t_name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`t_id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;

CREATE TABLE `t_idcard2` (
  `t_id` int(11) NOT NULL AUTO_INCREMENT,
  `t_number` varchar(255) DEFAULT NULL,
  `t_people_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`t_id`),
  KEY `FKrfj903v724ch0q2ex5u6m2min` (`t_people_id`),
  CONSTRAINT `FKrfj903v724ch0q2ex5u6m2min` FOREIGN KEY (`t_people_id`)
REFERENCES `t_people2` (`t_id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
```

3.一对多（单向）

1.主要实体

```
Student -> t_student
@Id
    @Column(name = "t_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
-----------------------------------
Project -> t_project

@ManyToOne()
    @JoinColumn(name = "t_student_id")
    private Student student;
#### 1
加了级联
测试一个student
多个project ,设置同一个student
会报错
```

```
解决方法
不用级联
先保存student，再保存project

#### 2
尝试加级联看看怎么才能可以同时插入多条project(student为同一个
解决方案
需要在test方法上加上下面连个注解
作用是使得代码在同一个事务中，同时自动提交
没有@Commit ,则不会提交会回滚
@Transactional
@Commit
```

2.自动生成的表

```
CREATE TABLE `t_student` (
  `t_id` int(11) NOT NULL AUTO_INCREMENT,
  `t_name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`t_id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;

CREATE TABLE `t_project` (
  `t_id` int(11) NOT NULL AUTO_INCREMENT,
  `t_sub_name` varchar(255) DEFAULT NULL,
  `t_student_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`t_id`),
  KEY `FKjbi0sj2aqjxfmj2tki0wv33xd` (`t_student_id`),
  CONSTRAINT `FKjbi0sj2aqjxfmj2tki0wv33xd` FOREIGN KEY (`t_student_id`)
REFERENCES `t_student` (`t_id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8;
```

4，一对多（双向）

1.主要实体

```
Student2 -> t_student2
@Id
    @Column(name = "t_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "t_name")
    private String name;

    @OneToMany(mappedBy = "student2")//维护外键的一方对应的实体类中的本类类型的属性字段
名称
    private List<Project2> project2;

Project2 -> t_project2
```

```
@Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "t_id")
    private Integer id;

    @Column(name = "t_sub_name")
    private String name;

    @ManyToOne()
    @JoinColumn(name = "t_student_id")
    private Student2 student2;
```

2.自动生成的表结构

```
CREATE TABLE `t_student2` (
  `t_id` int(11) NOT NULL AUTO_INCREMENT,
  `t_name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`t_id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;

CREATE TABLE `t_project2` (
  `t_id` int(11) NOT NULL AUTO_INCREMENT,
  `t_sub_name` varchar(255) DEFAULT NULL,
  `t_student_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`t_id`),
  KEY `FKl48vus0ax7ywxcex420rik40` (`t_student_id`),
  CONSTRAINT `FKl48vus0ax7ywxcex420rik40` FOREIGN KEY (`t_student_id`)
REFERENCES `t_student2` (`t_id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;
```

5.多对多（单向）

```
@ManyToMany
作用：用于映射多对多关系
属性：
cascade：配置级联操作。
fetch：配置是否采用延迟加载。
targetEntity：配置目标的实体类。映射多对多的时候不用写。

@JoinTable
作用：针对中间表的配置
属性：
nam：配置中间表的名称
joinColumns：中间表的外键字段关联当前实体类所对应表的主键字段
inverseJoinColumn：中间表的外键字段关联对方表的主键字段

@JoinColumn
作用：用于定义主键字段和外键字段的对应关系。
属性：
name：指定外键字段的名称
```

```
referencedColumnName：指定引用主表的主键字段名称
unique：是否唯一。默认值不唯一
nullable：是否允许为空。默认值允许。
insertable：是否允许插入。默认值允许。
updatable：是否允许更新。默认值允许。
columnDefinition：列的定义信息。
```

1.主要实体

```
@Id
    @Column(name = "t_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "t_name")
    private String name;
--------------------------------------------------------------


@Id
    @Column(name = "t_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "t_name")
    private String name;

    @ManyToMany(cascade = CascadeType.PERSIST)
    @JoinTable(name="user_role_rel",//中间表的名称
            //中间表user_role_rel字段关联sys_role表的主键字段role_id
            joinColumns=
{@JoinColumn(name="role_id",referencedColumnName="t_id")},
            //中间表user_role_rel的字段关联sys_user表的主键user_id
            inverseJoinColumns=
{@JoinColumn(name="user_id",referencedColumnName="t_id")}
    )
    private List<SysUser> sysUserList;

###
```

2.自动生成的表结构

```
CREATE TABLE `t_sys_user` (
  `t_id` int(11) NOT NULL AUTO_INCREMENT,
  `t_name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`t_id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8;


CREATE TABLE `t_sys_role` (
  `t_id` int(11) NOT NULL AUTO_INCREMENT,
  `t_name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`t_id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;


CREATE TABLE `user_role_rel` (
  `role_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  KEY `FKm28g58dhcs5u9asuuww8ui43w` (`user_id`),
  KEY `FKi2omtqgkldjbgukc3ry5hsdf` (`role_id`),
  CONSTRAINT `FKi2omtqgkldjbgukc3ry5hsdf` FOREIGN KEY (`role_id`) REFERENCES
`t_sys_role` (`t_id`),
  CONSTRAINT `FKm28g58dhcs5u9asuuww8ui43w` FOREIGN KEY (`user_id`) REFERENCES
`t_sys_user` (`t_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

6.多对多（多向）

1.主要的实体类

```
@Id
    @Column(name = "t_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "t_name")
    private String name;

    @ManyToMany(mappedBy = "sysUserList2")//维护外键的表的对应实体中的属性字段名
    private List<SysRole2> sysRole2;


@Id
    @Column(name = "t_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "t_name")
    private String name;

    @ManyToMany(cascade = CascadeType.PERSIST)
    @JoinTable(name="user_role_rel2",//中间表的名称
            //中间表user_role_rel字段关联sys_role表的主键字段role_id
            joinColumns=
{@JoinColumn(name="role_id",referencedColumnName="t_id")},
            //中间表user_role_rel的字段关联sys_user表的主键user_id
            inverseJoinColumns=
{@JoinColumn(name="user_id",referencedColumnName="t_id")}
    )
    private List<SysUser2> sysUserList2;
```

2.自动生成的表结构

```
CREATE TABLE `t_sys_user2` (
  `t_id` int(11) NOT NULL AUTO_INCREMENT,
  `t_name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`t_id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8;


CREATE TABLE `t_sys_role2` (
  `t_id` int(11) NOT NULL AUTO_INCREMENT,
  `t_name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`t_id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8;


CREATE TABLE `user_role_rel2` (
  `role_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  KEY `FK5o1if0v99f02hwcnvmitah1j7` (`user_id`),
  KEY `FKkjme7hdhxcgkbka7kox6r9ul` (`role_id`),
  CONSTRAINT `FK5o1if0v99f02hwcnvmitah1j7` FOREIGN KEY (`user_id`) REFERENCES
`t_sys_user2` (`t_id`),
  CONSTRAINT `FKkjme7hdhxcgkbka7kox6r9ul` FOREIGN KEY (`role_id`) REFERENCES
`t_sys_role2` (`t_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

2.联合主键

1.方式一

```
1.通过 主键实体打上注解@Embeddabl
表实体中主键属性打上注解 @EmbeddedId

2.主要的实体
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Embeddable
public class ComputerPK implements Serializable {

    @Column(name = "t_ip")
    private String ip;

    @Column(name = "t_owner_id")
    private String ownerId;


}


@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Entity
```

```
@Table(name = "t_computer")
public class Computer {

    @EmbeddedId
    private ComputerPK computerPK;

    @Column(name="t_brand_name")
    private String brandName;

}
3.自动生成的表结构
CREATE TABLE `t_computer` (
  `t_ip` varchar(255) NOT NULL,
  `t_owner_id` varchar(255) NOT NULL,
  `t_brand_name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`t_ip`,`t_owner_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

2.方式二

```
1.@IdClass 配合 @Id  和上面的效果差不多，可能根据方法名字操作方便点

2.主要的实体
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Entity
@Table(name = "t_computer2")
@IdClass(Computer2PK.class)
public class Computer2 {

    @Id
    @Column(name = "t_ip")
    private String ip;

    @Id
    @Column(name = "t_owner_id")
    private String ownerId;

    @Column(name="t_brand_name")
    private String brandName;

}
--------------------------------------------------
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class Computer2PK implements Serializable {

    private String ip;

    private String ownerId;


}
```

```
3.自动生成的表结构

CREATE TABLE `t_computer2` (
  `t_ip` varchar(255) NOT NULL,
  `t_owner_id` varchar(255) NOT NULL,
  `t_brand_name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`t_ip`,`t_owner_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

3.审计

1.添加注解

```
1.启动类

@EnableJpaAuditing
2.注解类
@EntityListeners({AuditingEntityListener.class})
```

2.自定义实现接口AuditorAware的类

```
@Component
public class AuditConfig implements AuditorAware {
    /**
     * Returns the current auditor of the application.
     *
     * @return the current auditor
     */
    @Override
    public Optional getCurrentAuditor() {
        return Optional.of("allen");
    }
}
```

3.在实体类字段加上注解

```
@CreatedBy ： 由谁创建这条记录
@LastModifiedBy：是谁最后更新了这条记录
@CreatedDate：创建时间
@LastModifiedDate：最后更新时间
```

1668440387641

4.更新时创建时间和人为空的解决

```
1.审计时，更新操作，创建人和创建时间为空
解决方法updatable = false
@Column(name = "cre_date", updatable = false)
```

5.自定义的sql审计是不会起作用的

```
@Modifying
    @Transactional
    @Query(nativeQuery = true,
    value = "update t_my_audit t set t.t_name=:#{#req.name} where t.t_id=:#
{#req.id}")
    int updateMyAudit(@Param("req") MyAudit myAudit);


@Test
    public void updateTest02() {
        MyAudit myAudit = new MyAudit();
        myAudit.setId(1);
        myAudit.setName("frank");
        int i = myAuditRepository.updateMyAudit(myAudit);
        System.out.println(i);
        /**
         * Hibernate: update t_my_audit t set t.t_name=? where t.t_id=?
         * 1
         */

    }
```

6@pre

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Entity
@Table(name = "t_my_audit3")
public class MyAudit3 {

    @Id
    @Column(name = "t_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "t_name")
    private String name;


    @Column(name = "cre_user",updatable = false)
    private String createUser;

    @Column(name = "modi_user")
    private String modifyUser;


    @Column(name = "cre_date",updatable = false)
    private LocalDateTime createDate;

    @Column(name = "modi_date")
    private LocalDateTime modifyDate;


    @PrePersist
    public void insert() {
        this.createDate = this.modifyDate = LocalDateTime.now();
        this.createUser = this.modifyUser ="mark";
    }

    @PreUpdate
    public void update() {
        this.modifyDate = LocalDateTime.now();
        this.modifyUser = "mark";
    }
}
```

4.jpa继承

1.SINGLE_TABLE

1.简单说明

```
单表继承策略 SINGLE_TABLE

父类实体和子类实体共用一张数据库表，在表中通过一个辨别字段的值来区分不同类别的实体。
```

2表对应的实体

1.父类

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)//继承的策略
@Table(name = "WINDOW_FILE")
@DiscriminatorColumn(name = "DISCRIMINATOR", discriminatorType =
DiscriminatorType.STRING, length = 30)  // 指定辨别字段的类型为String, 长度30
@DiscriminatorValue("WindowFile")//指定辨别的字段值
public class WindowFile {

     @Id
     @GeneratedValue(strategy = GenerationType.AUTO)
     private Integer id;

     @Basic
     @Column(name = "NAME")
     private String name;

     @Basic
```

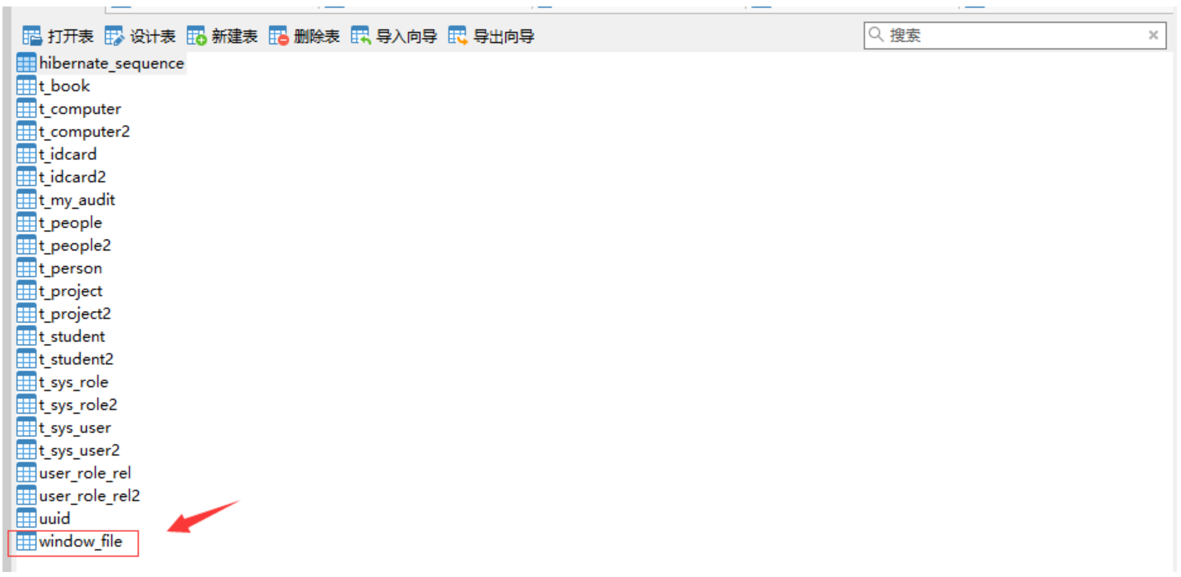
```
    @Column(name = "TYPE")
    private String type;

    @Basic
    @Column(name = "DATE")
    private Date date;

}
```

2.子类1

```
@Entity
@DiscriminatorValue("Document")
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class Document extends WindowFile {

 @Basic
 @Column(name = "SIZE")
 private String size;

}
```

3.子类2

```
@Entity
@DiscriminatorValue("Folder")
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class Folder extends WindowFile {

 @Basic
 @Column(name = "FILE_COUNT")
 private Integer fileCount;

}
```



hibernate_sequence
t_book
t_computer
t_computer2
t_idcard
t_idcard2
t_my_audit
t_people
t_people2
t_person
t_project
t_project2
t_student
t_student2
t_sys_role
t_sys_role2
t_sys_user
t_sys_user2
user_role_rel
user_role_rel2
uuid
window_file

```
CREATE TABLE `window_file` (
  `discriminator` varchar(30) NOT NULL,
  `id` int(11) NOT NULL,
  `date` datetime(6) DEFAULT NULL,
  `name` varchar(255) DEFAULT NULL,
  `type` varchar(255) DEFAULT NULL,
  `size` varchar(255) DEFAULT NULL,
  `file_count` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

2.JOINED

1.简单说明

```
父类实体和子类实体分别对应数据库中不同的表，父类定义的内容为子类们公共的属性，子类实体中定义的内容为扩展的属性。
实际生成的表结构如下：

表：T_ANIMAL，字段：   ID,COLOR,NAME

表：T_BIRD   ，字段：  SPEED,ID(既是外键，也是主键)

表：T_DOG，字段：    LEGS,ID(既是外键，也是主键)
```

2.表对应实体

1.父类

```java
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;

@Entity
@Table(name = "t_animal")
@Inheritance(strategy = InheritanceType.JOINED)
@DiscriminatorColumn(name = "aaa")  // 辨别字段 AAA
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Data
public class Animal {

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;

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

    @Column(name = "color")
    private String color;
}
```

2.子类1

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Column;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity
@Table(name = "t_bird")
@DiscriminatorValue("bird")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Bird extends Animal {

    @Column(name = "speed")
    private String speed;

    @Override
    public String toString() {
        return super.toString() + "Bird{" +
                "speed='" + speed + '\'' +
                '}';
    }
}
```

3.子类2

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Column;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity
@Table(name = "t_dog")
@DiscriminatorValue("dog")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Dog extends Animal {

    @Column(name = "legs")
    private Integer legs;

    @Override
    public String toString() {
        return super.toString() + "Dog{" +
                "legs=" + legs +
                '}';
    }
}
```

3.自动生成的表结构

1.父表（公共表）

```
CREATE TABLE `t_animal` (
  `aaa` varchar(31) NOT NULL,
  `id` int(11) NOT NULL,
  `color` varchar(255) DEFAULT NULL,
  `name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

2.子表1

```

CREATE TABLE `t_bird` (
  `speed` varchar(255) DEFAULT NULL,
  `id` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `FKky0iakih6f0xm2eqtq3p5s8u7` FOREIGN KEY (`id`) REFERENCES
`t_animal` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

3.子表2

```
CREATE TABLE `t_bird` (
  `speed` varchar(255) DEFAULT NULL,
  `id` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `FKky0iakih6f0xm2eqtq3p5s8u7` FOREIGN KEY (`id`) REFERENCES
`t_animal` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

3.Table_pre_class

1.简单说明

```
父类实体和子类实体各自生成表，实体对应自己生成的表，子类实体对应的表的字段保存所有的属性，包括从
父类实体中继承的属性

一旦使用这种策略就意味着你不能使用AUTO generator 和IDENTITY generator，即主键值不能采用数
据库自动生成
```

2.表对应的实体类

1.父类

```
@Data
@AllArgsConstructor
@NoArgsConstructor

@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
@Table(name = "t_vehicle")
public class Vehicle {

    @Id
    //@GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;

    @Column(name = "SPEED")
    private Integer speed;// 速度

}
```

2.子类1

```
@Data
@AllArgsConstructor
@NoArgsConstructor

@Entity
@Table(name = "t_car")
public class Car extends Vehicle {

    @Column(name = "engine")
    private String engine;// 发动机

}
```

3.自动生成的表结构

1.父表

```
CREATE TABLE `t_vehicle` (
  `id` int(11) NOT NULL,
  `speed` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

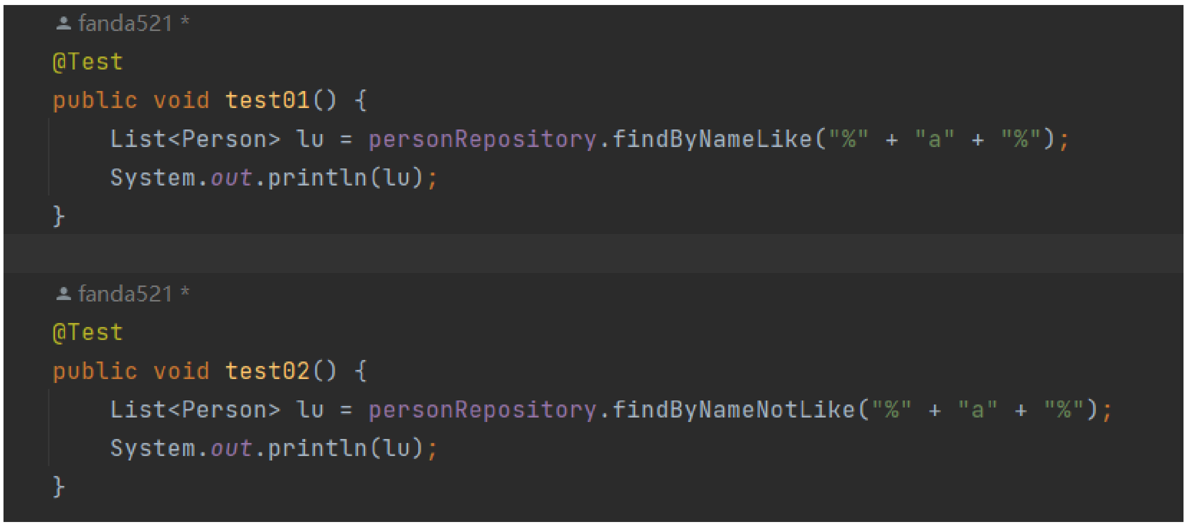
2.字表1

```
CREATE TABLE `t_car` (
  `id` int(11) NOT NULL,
  `speed` int(11) DEFAULT NULL,
  `engine` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

5.常见的坑

1.模糊查询

```
1.根据方法名的方式
需要在参数值那里手动加上 %name%
2.@query方式也需要加上 %name%
3.entityManager
这种的也需要在参数值那里加上
```



```
fanda521 *
@Test
public void test01() {
    List<Person> lu = personRepository.findByNameLike("%" + "a" + "%");
    System.out.println(lu);
}

fanda521 *
@Test
public void test02() {
    List<Person> lu = personRepository.findByNameNotLike("%" + "a" + "%");
    System.out.println(lu);
}
```

```
-- jpa
@Query(value = "from Person t where t.name like :name")
    List<Person> getNameLike(String name);


@Test
    public void testLikeQuery() {
        List<Person> nameLike = personRepository.getNameLike("%" + "a" + "%");
        System.out.println(nameLike);
    }

-- native
@Query(nativeQuery = true,
    value = "select * from t_person t where t.t_name like :name")
    List<Person> getNameLikeNative(String name);

@Test
    public void testLikeQueryNative() {
        List<Person> nameLike = personRepository.getNameLikeNative("%" + "a" +
"%");
        System.out.println(nameLike);
    }
```

```
public List<Person> getByNameLikeSql(String name) {
        String sql =" select t.t_id id,t.t_name name , \n" +
                " t.t_age age,t.t_address address, t.t_birthday birthday \n" +
                " from t_person t \n" +
                " where t.t_name like :name";

        Query nativeQuery = entityManager.createNativeQuery(sql);
        nativeQuery.setParameter("name","%" + name + "%");
        org.hibernate.query.Query query =
nativeQuery.unwrap(org.hibernate.query.Query.class).setResultTransformer(Transfo
rmers.aliasToBean(Person.class));
        List<Person> resultList = query.getResultList();
        return resultList;
    }

@Test
    public void testLikeEntityManagerNative() {
        List<Person> a = personService.getByNameLikeSql("a");
        System.out.println(a);
    }
```

2.传参

1.?1

```
通过位置传参
是从1开始的
-- 1.jpa
@Query("from Person where name=?1 or age =?2")
    List<Person> selectByNameOrAge(String name,Integer age);

-- 2. native
@Query(value = "select t_id as id,t_name as name ,t_age as age,t_address as
address,t_birthday as birthday from t_person where t_name=?1 or age =?
2",nativeQuery = true)
    List<Person> selectByNameOrAgeNative(String name,Integer age);
--3.entityManager
public List<Person> getByNameLikeSqlIndex(String name) {
        String sql =" select t.t_id id,t.t_name name , \n" +
                " t.t_age age,t.t_address address, t.t_birthday birthday \n" +
                " from t_person t \n" +
                " where t.t_name like ?1";

        Query nativeQuery = entityManager.createNativeQuery(sql);
        nativeQuery.setParameter(1,"%" + name + "%");
        org.hibernate.query.Query query =
nativeQuery.unwrap(org.hibernate.query.Query.class).setResultTransformer(Transfo
rmers.aliasToBean(Person.class));
        List<Person> resultList = query.getResultList();
        return resultList;
    }
```

2.:name

```
通过参数名传参
--1.jpa
@Query("from Person where name=:name")
    List<Person> selectByName(@Param("name") String name);
-- 2.native
@Query(value = "select t_id ,t_name  ,t_age ,t_address ,t_birthday  from
t_person t where t_name=:name",nativeQuery = true)
    List<Person> selectByNameNative(@Param("name") String name);
-- 3.entityManager
public List<Person> getByNameLikeSql(String name) {
        String sql =" select t.t_id id,t.t_name name , \n" +
                " t.t_age age,t.t_address address, t.t_birthday birthday \n" +
                " from t_person t \n" +
                " where t.t_name like :name";

        Query nativeQuery = entityManager.createNativeQuery(sql);
        nativeQuery.setParameter("name","%" + name + "%");
        org.hibernate.query.Query query =
nativeQuery.unwrap(org.hibernate.query.Query.class).setResultTransformer(Transfo
rmers.aliasToBean(Person.class));
        List<Person> resultList = query.getResultList();
        return resultList;
    }
```

3.结果接收

1.使用vo（@query失败 jpa）

```
1.不使用表对应的实体类接收是不行的。会报错
org.springframework.core.convert.ConverterNotFoundException: No converter found
capable of converting from type
[org.springframework.data.jpa.repository.query.AbstractJpaQuery$TupleConverter$T
upleBackedMap] to type [com.wang.example.springbootdatajpa.entity.PersonVo]
```

2.使用vo（@query失败 native）

```
1.不使用表对应的实体类接收是不行的。会报错
org.springframework.core.convert.ConverterNotFoundException: No converter found
capable of converting from type
[org.springframework.data.jpa.repository.query.AbstractJpaQuery$TupleConverter$T
upleBackedMap] to type [com.wang.example.springbootdatajpa.entity.PersonVo]
```

3.domain接收部分字段

```
1.使用表对应的实体类接收，但是只接收部分字段
2.结果是不行的，报错
/**
     * 用domain 接收部分字段
     * 结果：失败
     * 错误：org.springframework.core.convert.ConversionFailedException:
     * Failed to convert from type [java.lang.Object[]]
     * to type [@org.springframework.data.jpa.repository.Query
com.wang.example.springbootdatajpa.entity.Person]
     * for value '{allan, 1, 2022-11-30 10:40:05.0}';
     * nested exception is
org.springframework.core.convert.ConverterNotFoundException:
     * No converter found capable of converting from type [java.lang.String]
     * to type [@org.springframework.data.jpa.repository.Query
com.wang.example.springbootdatajpa.entity.Person]
     */
```

4.domain接收部分字段（native）

```
1.失败
/**
     * 用domain 接收部分字段(native)
     * 结果：失败
     * 错误：
     * 2022-12-29 11:56:45.258  WARN 37704 --- [           main]
o.h.engine.jdbc.spi.SqlExceptionHelper   :
     * SQL Error: 0, SQLState: S0022
     * 2022-12-29 11:56:45.259 ERROR 37704 --- [           main]
o.h.engine.jdbc.spi.SqlExceptionHelper   :
     * Column 't_id' not found.
     *
     * org.springframework.dao.InvalidDataAccessResourceUsageException:
     * could not execute query; SQL [select t.t_name name,t.t_address
address,t.t_birthday birthday from t_person t];
     * nested exception is org.hibernate.exception.SQLGrammarException: could
not execute query
     */
```