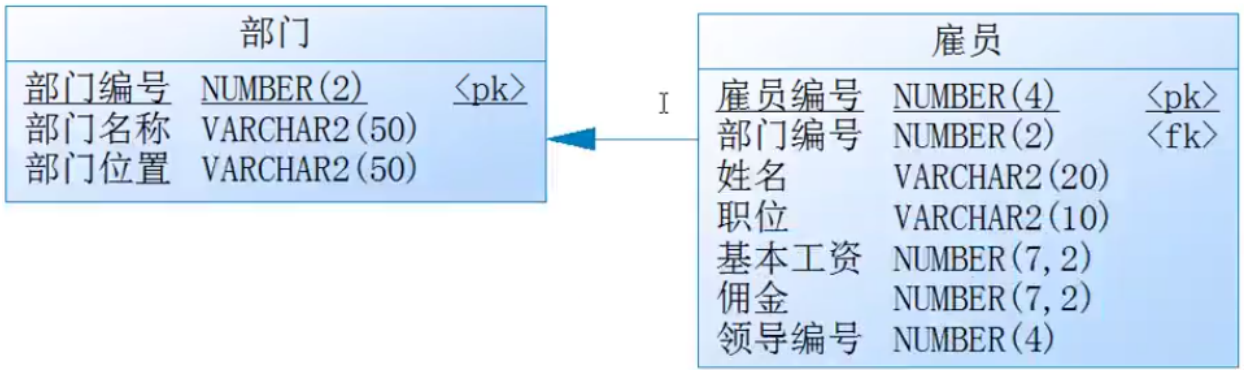
2、具体内容

简单Java类时现在面向对象设计的主要分析基础，但是对于设计的开发之中简单Java类的定义来源是有依据的，往往都是根据数据表的结构来实现简单Java类。

在数据库之中实际上是提供有若干个数据表的，那么每一张实体数据表实际上都可以描述出一些具体的事物概念，例如：雇员信息表、部门信息表一看就知知道描述的是雇员或部门的信息。

那么按照这个思路回到程序之中你会发现，程序类的定义形式实际上和这些实体表的差别并不大，所以在实际的开发项目之中数据表以简单Java类之间的基本映射关系如下：

- 数据实体表设计 = 类的定义；
- 表中的字段 = 类的成员属性；
- 表中外键关联 = 引用关联；
- 表的一行记录 = 累的一个实例化对象；
- 表的多行记录 = 对象数组；



在以上所对应数据表的关系之中可以发现有如下的关联存在：

- 一个部门有多个雇员；
- 一个雇员属于一个部门；
- 一个雇员有一个领导；

下面将以上的数据表转为简单java类的定义形式，在整体的程序代码之中要求可以获得如下信息：

- 根据部门信息获得以下内容：

  |- 一个部门的完整信息；
  |- 一个部门之中所有雇员的完整信息；
  |- 一个雇员对应的领导的信息；

- 根据雇员信息获得以下内容：

  |- 一个雇员所在部门信息；
  |- 一个雇员对应的领导信息；

对于数据表与简单java类之间的映射最好的解决步骤：先抛开所有的关联字段不看，写出类的基本组成，而后在通过引用配置关联字段的关系。

第一步：分别定义Emp、Dept两个实体类

```
class Dept{
	private long deptno;
	private String dname;
	private String loc;
	public Dept(long deptno, String dname, String loc) {
		this.deptno = deptno;
		this.dname = dname;
		this.loc = loc;
	}
	//setter、getter无参构造略
	public String getInfo() {
		return "【部门信息】 [部门编号=" + deptno + ", 部门名称=" + dname + ", 部门编号
=" + loc + "]";
	}
}

class Emp{
	private long empno;
	private String ename;
	private String job;
	private double sal;
	private double comm;
	public Emp(long empno, String ename, String job, double sal, double comm) {
		super();
		this.empno = empno;
		this.ename = ename;
		this.job = job;
		this.sal = sal;
		this.comm = comm;
	}
	//setter、getter无参构造略
	public String getInfo() {
```

```
        return "【雇员信息】 [雇员编号 =" + empno + ", 雇员姓名=" + ename + ", 雇员职位
=" + job + ", 基本工资=" + sal + ", 佣金=" + comm + "]";
    }
```

第二步：配置所有的关联字段

```
class Dept{
    private long deptno;
    private String dname;
    private String loc;
    private Emp emps[];//多个雇员信息
    public Emp[] getEmps() {
        return emps;
    }
    public void setEmps(Emp[] emps) {
        this.emps = emps;
    }
    public Dept(long deptno, String dname, String loc) {
        this.deptno = deptno;
        this.dname = dname;
        this.loc = loc;
    }
    //setter、getter无参构造略
    public String getInfo() {
        return "【部门信息】 [部门编号=" + deptno + ", 部门名称=" + dname + ", 部门编号
=" + loc + "]";
    }
}

class Emp{
    private long empno;
    private String ename;
    private String job;
    private double sal;
    private double comm;
    private Dept dept;//所属部门
    private Emp mgr;//所属

    public Dept getDept() {
        return dept;
    }
    public void setDept(Dept dept) {
        this.dept = dept;
    }
    public Emp getMgr() {
        return mgr;
    }
    public void setMgr(Emp mgr) {
        this.mgr = mgr;
    }
    public Emp(long empno, String ename, String job, double sal, double comm) {
        super();
        this.empno = empno;
```

```
            this.ename = ename;
            this.job = job;
            this.sal = sal;
            this.comm = comm;
        }
        //setter、getter无参构造略
        public String getInfo() {
            return "【雇员信息】 [雇员编号 =" + empno + ", 雇员姓名=" + ename + ", 雇员职位
=" + job + ", 基本工资=" + sal + ", 佣金=" + comm + "]";
        }

}
public class JavaDemo {
        public static void main(String[] args) {

        }
}
```

在以后进行项目开发的过程之中一定是分两个步骤实现的：

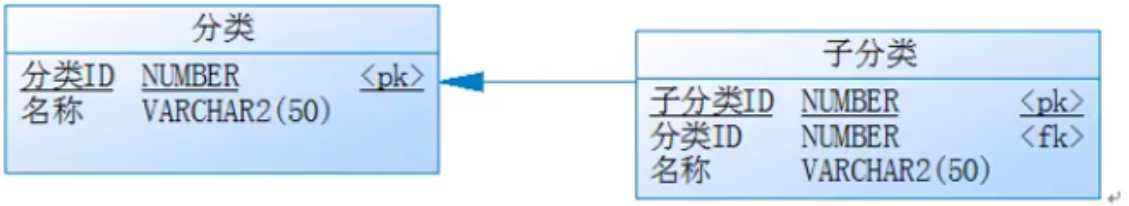
- 第一步：根据表的结构关系进行对象的配置；
- 第二步：根据要求通过结构获取数据；

范例：实现项目开发要求

```
public class JavaDemo {
        public static void main(String[] args) {
            //第一步：根据关系进行类的定义
            //定义出各个的实例化对象，此时并没有任何的关联定义
            Dept dept = new Dept(1,"财务部","shanghai");
            Emp empA = new Emp(7360L,"SMITH","CLERK",800.00,0.0);
            Emp empB = new Emp(7566,"FORD","MANAGER",24500.00,0.0);
            Emp empC = new Emp(7839L,"KING","PRESIDENT",5000.00,0.0);
            //需要为对象进行关联的设置
            empA.setDept(dept);//设置雇员与部门的关联
            empB.setDept(dept);//设置雇员与部门的关联
            empC.setDept(dept);//设置雇员与部门的关联
            empA.setMgr(empB);//设置雇员与部门的关联
            empB.setMgr(empC);//设置雇员与部门的关联
            dept.setEmps(new Emp[] {empA,empB,empC});//部门与雇员
            //第二步：根据关系获取数据
            System.out.println(dept.getInfo());//部门信息
            for(int x=0;x<dept.getEmps().length;x++){
                System.out.println("\t|-"+dept.getEmps()[x].getInfo());
                if(dept.getEmps()[x].getMgr()!=null) {
                System.out.println("\t\t|-"+dept.getEmps()[x].getMgr().getInfo());
                }
            }
            System.out.println("-----------------------------------");
            System.out.println(empB.getDept().getInfo());      //根据雇员获取部门信息
            System.out.println(empB.getMgr().getInfo());//根据雇员获取领导信息
        }
}
```

在以后的开发之中这种转换的定义形式一定是要求熟练完成的；

2、具体内容



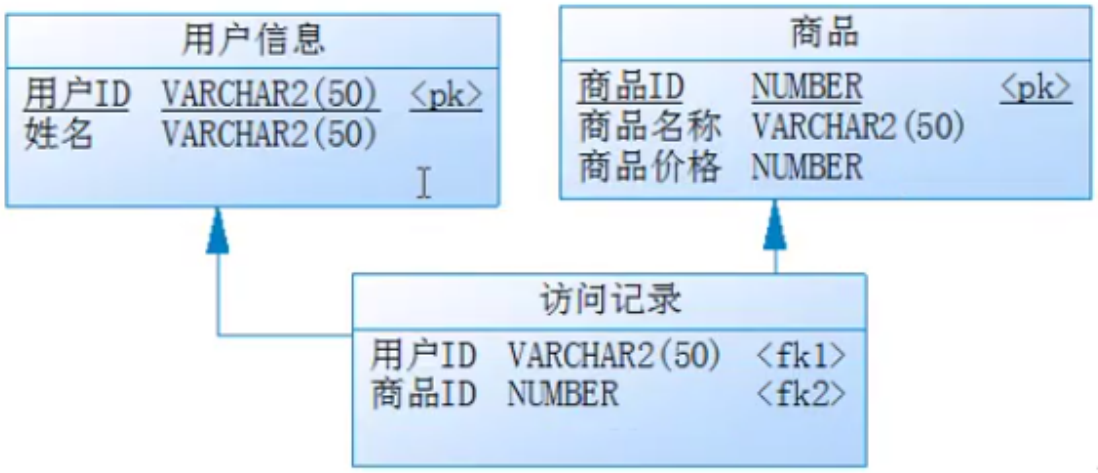
按照表的要求将表的结构转换为类结构，同时可以获取如下信息：

- 获取一个分类的完整信息；
- 可以根据分类获取其对应的所有子分类的信息。

```
class Item{
	private long iid;
	private String title;
	private Subitem subitems[];
	public Subitem[] getSubitems() {
		return subitems;
	}
	public void setSubitems(Subitem[] subitems) {
		this.subitems = subitems;
	}
	public Item() {
	}
	public Item(long iid, String title) {
		this.iid = iid;
		this.title = title;
	}
	public String getInfo() {
```

```
            return "【分类信息】 [iid=" + iid + ", title=" + title + "]";
        }

}
class Subitem{
        private long sid;
        private String title;
        private Item item;
        public Item getItem() {
                return item;
        }
        public void setItem(Item item) {
                this.item = item;
        }
        public Subitem() {
        }
        public Subitem(long sid, String title) {
                this.sid = sid;
                this.title = title;
        }

        public String getInfo() {
                return "【子分类信息】[sid=" + sid + ", title=" + title + "]";
        }

}
public class JavaDemo {
        public static void main(String[] args) {
                //第一步：根据结构设置对象数据
                Item item = new Item(1L,"tushu");
                Subitem subitems[] = new Subitem[] {
                        new Subitem(10L,"programbook"),
                        new Subitem(10L,"graphbook")
                };
                item.setSubitems(subitems);//一个分类下有多个子分类
                for(int x=0;x<subitems.length;x++) {
                        subitems[x].setItem(item);
                }
                //第二步：根据要求获取数据
                System.out.println(item.getInfo());
                for(int x=0;x<item.getSubitems().length;x++) {
                        System.out.println("\t|-"+item.getSubitems()[x].getInfo());
                }
        }
}
```

2、具体内容



将以上的结构转换为类结构，并且可以获取如下的信息：

- 获取一个用户访问的所有商品的详细信息；
- 获取一个商品被浏览过的全部的用户信息。

对于此时的程序只需要去考虑实体表的设计即可，也就是说对于中间的访问记录信息表不要求你进行转换处理，只定义两个类即可。

```
class Member{
	private String mid;
	private String name;
	private Product products[];
	public Member(String mid, String name) {
		this.mid = mid;
		this.name = name;
	}

	public Product[] getProducts() {
		return products;
	}
```

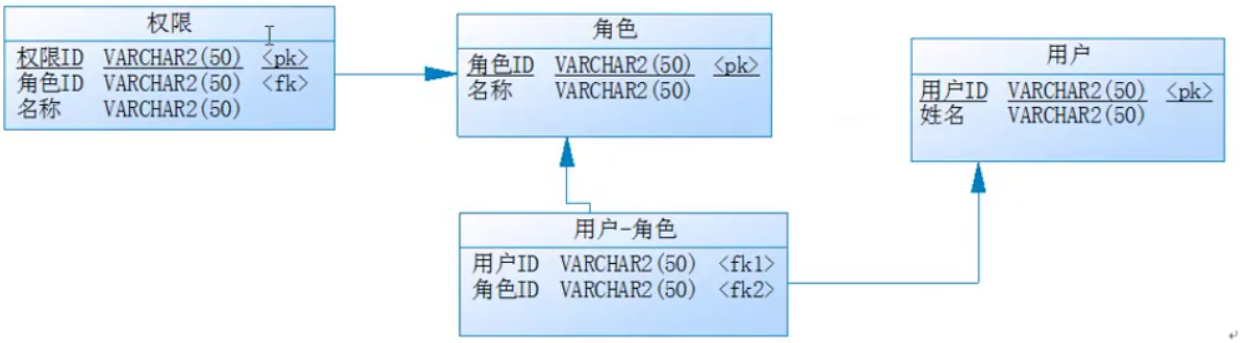
```
	public void setProducts(Product[] products) {
		this.products = products;
	}

	public String getInfo() {
		return "【用户信息】 [mid=" + mid + ", name=" + name + "]";
	}
}
class Product{
	private long pid;
	private String title;
	private double price;
	private Member members[];
	public Product(long pid, String title, double price) {
		super();
		this.pid = pid;
		this.title = title;
		this.price = price;
	}

	public Member[] getMembers() {
		return members;
	}

	public void setMembers(Member[] members) {
		this.members = members;
	}

	public String getInfo() {
		return "【商品信息】 [pid=" + pid + ", title=" + title + ", price=" + price + "]";
	}
}
public class JavaDemo {
	public static void main(String[] args) {
		//第一步：根据结构设置对象数据
		Member memA = new Member("mldn","sanzhang");
		Member memB = new Member("mldnjava","sili");
		Product proA = new Product(1L,"javaexploit",79.8);
		Product proB = new Product(2L,"bigheadset",2343.8);
		Product proC = new Product(3L,"xiaomiphone",8902.8);
		memA.setProducts(new Product[] {proA,proB,proC});
		memB.setProducts(new Product[] {proA});
		proA.setMembers(new Member[] {memA,memB});
		proB.setMembers(new Member[] {memA});
		proC.setMembers(new Member[] {memA});
		//第二步：根据要求获取数据
		System.out.println("-----------------根据用户查看浏览商品信息--------------------");
		System.out.println(memA.getInfo());
		for(int x=0;x<memA.getProducts().length;x++) {
			System.out.println("\t|-"+memA.getProducts()[x].getInfo());
		}
```

```
            System.out.println("----------------根据商品找到被浏览的记录-------------------");
            System.out.println(proA.getInfo());
            for(int x=0;x<proA.getMembers().length;x++) {
                System.out.println("\t|-"+proA.getMembers()[x].getInfo());
            }
        }
}
```

2、具体内容

在进行实际项目开发过程之中，对于用户的授权管理是一项重要的任务，下面给出了一个最为常见的用户权限管理的表结构设计，基本关系如下：

- 一个用户可以拥有多个角色，一个角色可能有多个用户；
- 一个角色可以拥有多个权限；



要求实现如下查询功能：

- 可以根据一个用户找到该用户对应的所有角色，以及每一个角色对应的所有权限信息；
- 可以根据一个角色找到该角色下的所有权限，以及拥有此角色的全部用户信息；
- 可以根据一个权限找到具备有此权限的所有用户信息。

```
class Member{
	private String mid;
	private String name;
	private Role roles[];
	public Role[] getRoles() {
		return roles;
	}
	public void setRoles(Role[] roles) {
		this.roles = roles;
	}
	public Member(String mid, String name) {
		super();
		this.mid = mid;
		this.name = name;
	}
	public String getInfo() {
		return "【用户信息】 [mid=" + mid + ", name=" + name + "]";
	}

}
class Role{
	private long rid;
	private String title;
	private Member members[];
	private Privilege privileges[];
	public Member[] getMembers() {
		return members;
	}
	public void setMembers(Member[] members) {
		this.members = members;
	}
	public Privilege[] getPrivileges() {
		return privileges;
	}
	public void setPrivileges(Privilege[] privileges) {
		this.privileges = privileges;
	}
	public Role(long rid, String title) {
		super();
		this.rid = rid;
		this.title = title;
	}
	public String getInfo() {
		return "【角色信息】 [rid=" + rid + ", title=" + title + "]";
	}

}
class Privilege{
	private long pid;
	private String title;
	private Role role;
	public Role getRole() {
```

```
		return role;
	}
	public void setRole(Role role) {
		this.role = role;
	}
	public Privilege(long pid, String title) {
		super();
		this.pid = pid;
		this.title = title;
	}
	public String getInfo() {
		return "【权限信息】 [pid=" + pid + ", title=" + title + "]";
	}
}

public class JavaDemo {
	public static void main(String[] args) {
		//第一步：根据结构设置对象数据
		Member memA = new Member("mlda","sili");
		Member memB = new Member("mldb","sanzhang");
		Role roleA = new Role(1L,"系统配置");
		Role roleB = new Role(2L,"备份管理");
		Role roleC = new Role(3L,"人事管理");
		Privilege priA = new Privilege(1000L,"系统初始化");
		Privilege priB = new Privilege(1001L,"系统系统还原");
		Privilege priC = new Privilege(1002L,"系统环境修改");
		Privilege priD = new Privilege(2000L,"备份员工数据");
		Privilege priE = new Privilege(1001L,"备份部门数据");
		Privilege priF = new Privilege(2002L,"备份公文数据");
		Privilege priG = new Privilege(3000L,"增加员工");
		Privilege priH = new Privilege(3001L,"编辑员工");
		Privilege priI = new Privilege(3002L,"浏览员工");
		Privilege priJ = new Privilege(3003L,"员工离职");
		//增加角色与权限的对应关系
		roleA.setPrivileges(new Privilege[] {priA,priB,priC});
		roleB.setPrivileges(new Privilege[] {priD,priE,priF});
		roleC.setPrivileges(new Privilege[] {priG,priH,priI,priJ});
		//增加权限与角色对应
		priA.setRole(roleA);
		priB.setRole(roleA);
		priC.setRole(roleA);
		priD.setRole(roleB);
		priE.setRole(roleB);
		priF.setRole(roleB);
		priG.setRole(roleC);
		priH.setRole(roleC);
		priI.setRole(roleC);
		priJ.setRole(roleC);
		//增加用户与角色的对应关系
		memA.setRoles(new Role[] {roleA,roleB});
		memB.setRoles(new Role[] {roleA,roleB,roleC});
		roleA.setMembers(new Member[] {memA,memB});
```

```
            roleB.setMembers(new Member[] {memA,memB});
            roleC.setMembers(new Member[] {memB});
            //第二步：根据要求获取数据
            System.out.println("------通过用户查找信息------");
            System.out.println(memB.getInfo());
            for(int x=0;x<memB.getRoles().length;x++) {
                System.out.println("\t|-"+memB.getRoles()[x].getInfo());
                for(int y=0;y<memB.getRoles()[x].getPrivileges().length;y++       ) {
                    System.out.println("\t\t|-"+memB.getRoles()[x].getPrivileges()
[y].getInfo());
                }
            }
            System.out.println("------通过角色查找信息------");
            System.out.println(roleB.getInfo());
            System.out.println("\t|- 浏览此角色下的所有权限信息:");
            for(int x=0;x<roleB.getPrivileges().length;x++) {
                System.out.println("\t\t|-"+roleB.getPrivileges()[x].getInfo());
            }
            System.out.println("\t|-浏览此角色下的所有用户信息:");
            for(int x=0;x<roleB.getMembers().length;x++) {
                System.out.println("\t\t|-"+roleB.getMembers()[x].getInfo());
            }
            System.out.println("------通过权限查找信息------");
            System.out.println(priA.getInfo());
            for(int x=0;x<priA.getRole().getMembers().length;x++) {
                System.out.println("\t|-"+priA.getRole().getMembers()[x].getInfo());
            }
        }
}
```