技术架构视图-数据持久化设计

胡协刚
软件架构师 **UML/RUP**专家

内容提要

- 对象持久化
- 数据模型的规范化
- 数据建模
- 支持数据模型的模式

对象持久化

持久化需求

- 业务中常常有大量的客户、供应商等等；有关每个人、每件事的细节信息都需要被收集和存放到某处
- 一个企业总是期望它的代表业务信息的数据或对象能够独立于使用它们的程序之外而存在
- 软件系统的状态从对它的一次调用到下一次的过程中被持久化，常常是很有价值的；这样当程序重启时，使得那些对象看起来一直都在那儿
- 多个应用可能需要使用同一对象
- 当一个对象希望与另外的对象交互时，不必要知道那个对象当前是在内存中，还是有待从持久存储中取出

什么是持久化

- 大多数有用的软件都将数据保存于持久化存储中：
  - 文件、数据库、磁带、CD等
- 传统的程序以一种专门规划好的方式来读写持久化的数据：
  - 对持久化存储读写数据的关注面与对问题域的关注面相隔离
- 面向对象的程序力图让存储和取回持久对象的工作尽可能地透明化
  - 对象的数据和行为，两者都可以被持久化
- 目前关系型数据库仍然是所有可用的工具中处理数据最快和最成熟的技术

什么是数据库

- 基于计算机的记录保持系统
- 被存储（持久化）数据的集合容器
- 数据模型（schema）
- 数据（统一和共享的）
- 硬件（物理考虑）
- 软件（数据库管理系统、客户端应用）
- 用户（数据库的开发人员、最终用户、DBA）
- 其它特性

数据集中控制的优势

- 减少冗余
- 减少不一致
- 数据共享
- 确保标准和命名惯用法
- 确保安全
- 完整性
- 并发性
- 平衡冲突的需求
- 数据独立于应用
- 由一个DBA管理员统一负责所有工作

数据库管理系统

- 扁平的文件

–每个数据条目一个记录

–用于简单的数据，没有事务处理需求

- 前关系型（层次型1965、网络型1970）
- 关系型（Ted Codd, 1980）
- 对象数据库（1990～现在）
- 扩展关系型（支持对象的关系型数据库）
- 其它

保证数据库的完整性

- 主键Primary key ：
  - 能够无二义地定位一个表中每一行的一个或多个属性值的组合
  - 每个表必须有一个主键
- 外键Foreign key ：
  - 一个表的主键在其它表（或其自身）中充当一个属性
- 引用的完整性Referential integrity:
  - 关系数据库管理系统必须确保外键与相应的主键保持一致，这可能成为数据库维护工作的一种重大挑战；很容易出现从一个表中删除一行，而造成其它某个表的外键指向了一个不再存在的行

事务处理

- ACID特性:
  - 原子性（不可分的工作单位）
  - 一致性（总是保持系统处于稳定状态）
  - 隔离性（避免并发执行的其它事务的影响）
  - 持久性（完成提交后，影响将被永久保持）

实例：事务处理

- 客户下一个订单（同时还注册了一个帐户）的UseCase:

✓ Add the customer information to the Customer table.

✓ Add address information to the Address table.

✓ Add phone information to the Phone table.

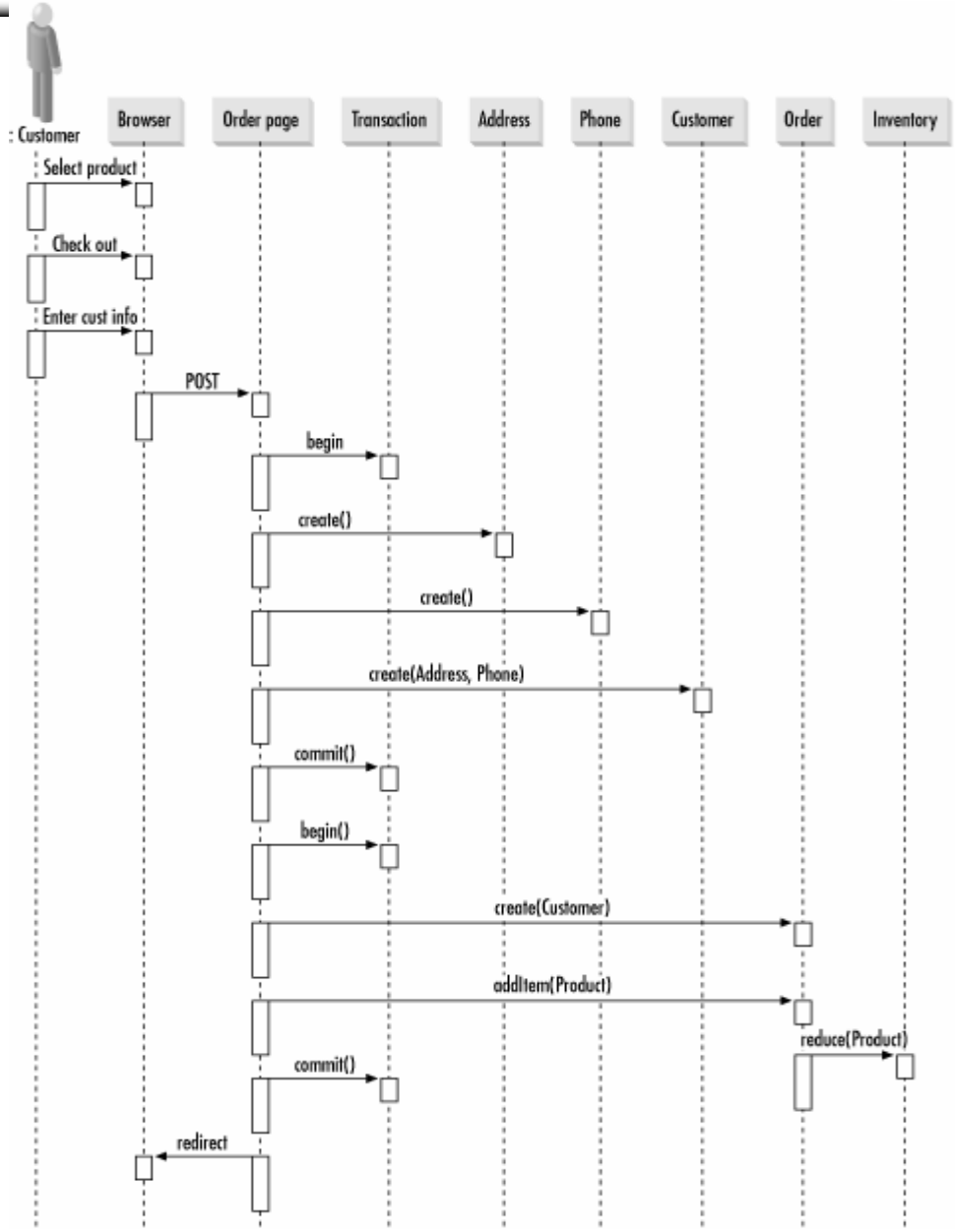
✓ Create an order in the Order table.

✓ Add a line item in the LineItem table.

✓ Decrease the inventory count in the Product table.

实例：事务处理

是将整个UseCase都封装在一个事务中，还是分成“增加帐户”与“下订单”两个事务？



常见的持久化实现途径

- 大部分应用还是选用关系型数据库
- 基于一个简单概念：表table
- 四种主要成分
  - 数据在表中予以呈现
  - 操作表的算子
  - 针对表的完整性规则
  - 表间的关联
- 对特定数据库所做的设计称为纲要图schema
- 非常成熟的技术
- 在所有的平台上可用
- 有范围很广的健壮工具可用

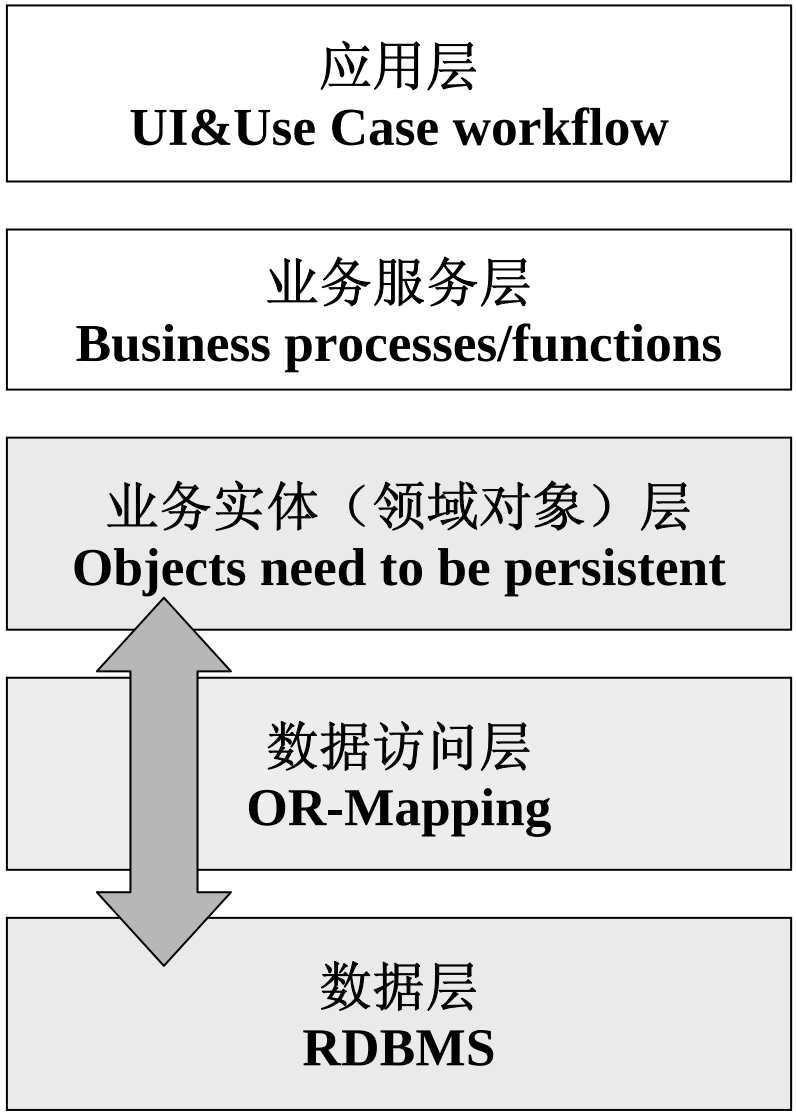
常见的持久化实现途径

- 对象同时具有数据（结构）和功能（行为）
  - 实践上，数据通常都被永久保存
  - 功能则由执行代码来提供
  - 在数据之外保存功能有利于向后兼容
- 目前面向对象的数据库已经可用
  - 不如扁平文件和关系数据库那样常用
  - 是最容易的一种保存对象的途径
  - 性能极高
  - 还没有形成统一标准，这可能是许多企业选用关系数据库而非对象数据库的主要原因
  - 对于许都项目而言，这是正确的技术选择

对象—关系映射

- 如果面向对象软件系统中的持久化机制选用了关系型数据库，则面临着如何将对象转化存储到关系数据库的表中，和从表中取回（重新还原为）对象的重大挑战
- 对象到关系的映射Object-Relation Mapping:
  - 将需要持久化存储的对象、对象属性以及对象间关系分别用关系数据库中的表、行、列以及外键等机制来表达
  - 关系到对象的映射则是它的反向机制

典型的持久化应用构架

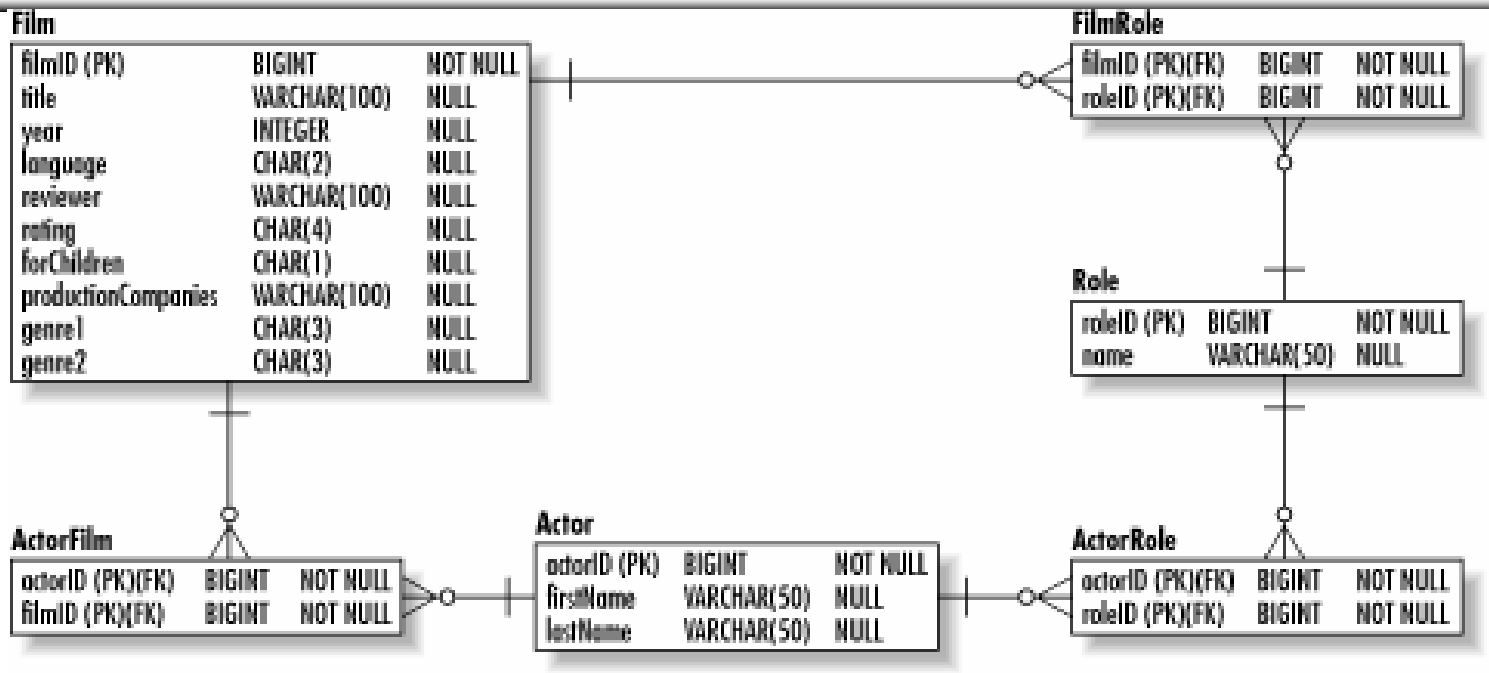


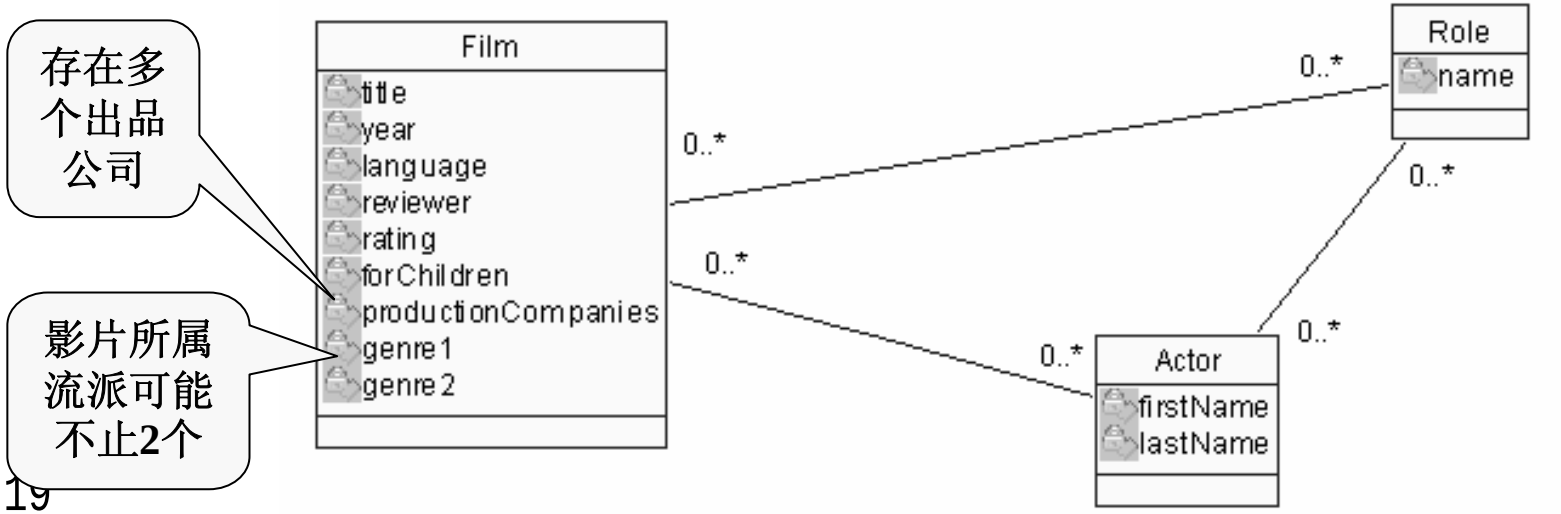
数据模型的规范化

数据库规范化Normalization的目标

- 清除冗余数据
  - 一个完全规范化的数据库将不存在除外键之外的任何重复内容
  - 去掉冗余数据使得数据库只保持最少的必要数据，对任何数据的维护只需访问一处，方便数据完整性的实现
- 确保关系模型的完整与一致
  - 规范化的过程迫使开发者检查数据模型的各个侧面，以确保关系模型的原则不被打破
- 提高伸缩性和适应性
  - 规范化的数据库适应应用变化的能力更强

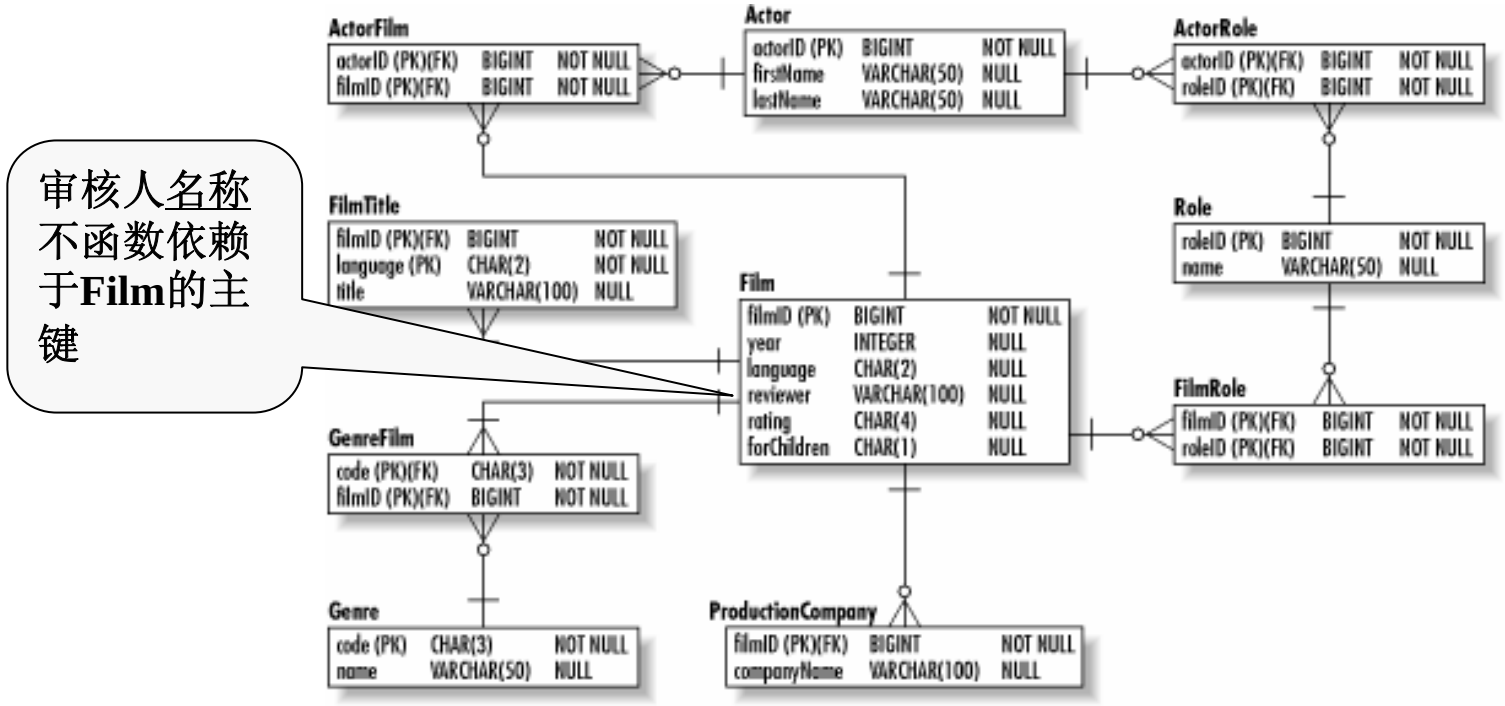
示例：违反第一范式1NF

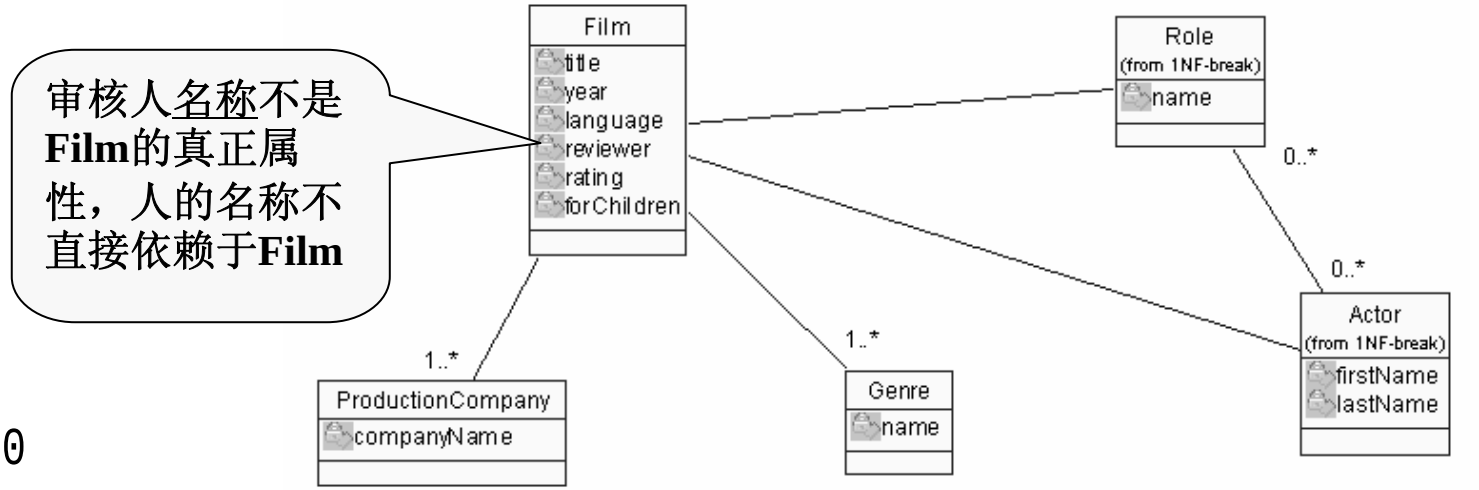




1NF:
实体所有属性的取值都应是单值的

示例：符合1NF 但仍违反2NF

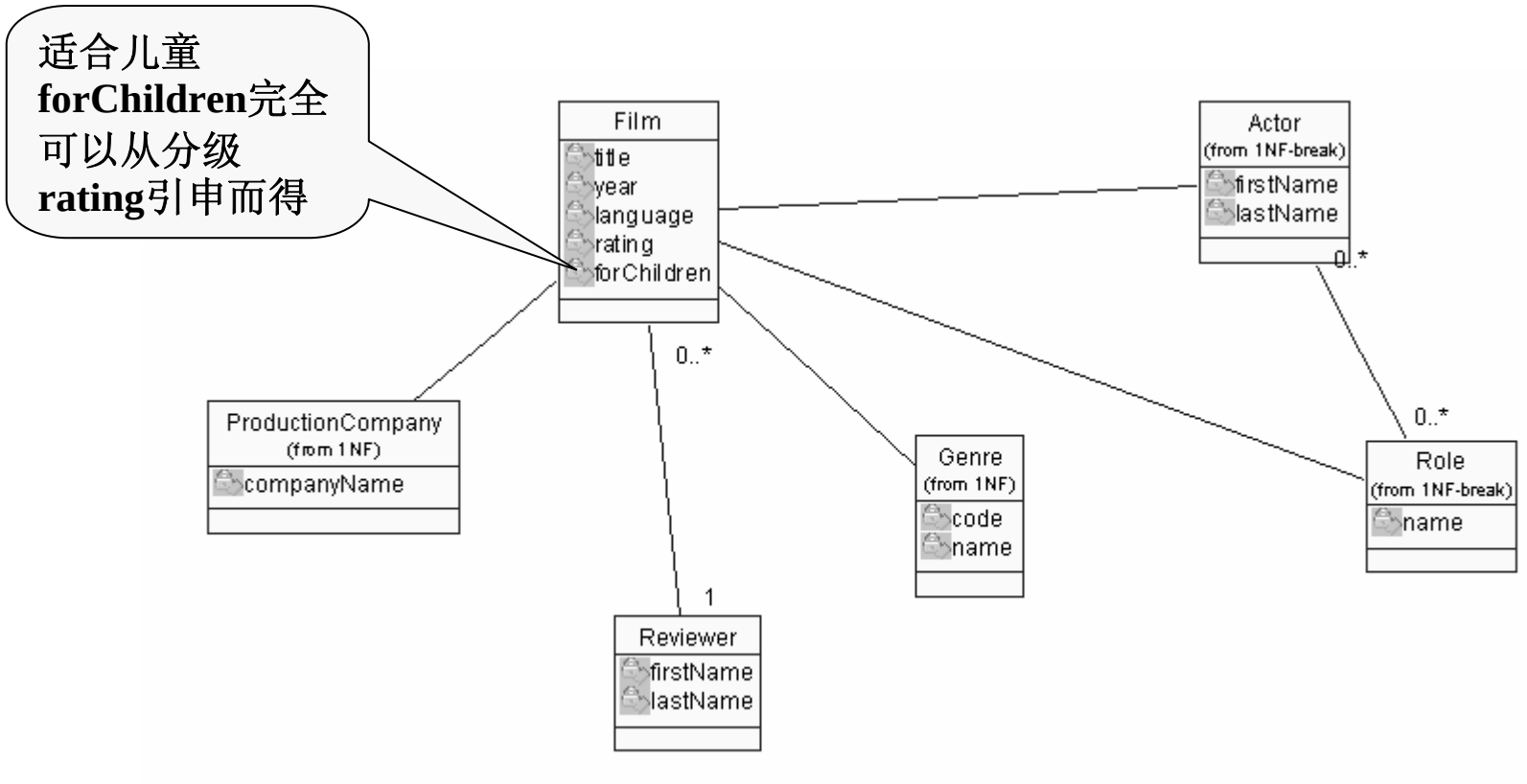




2NF:
实体所有非主键**non-key**属性都函数**functionally**依赖于其主键（属性组）

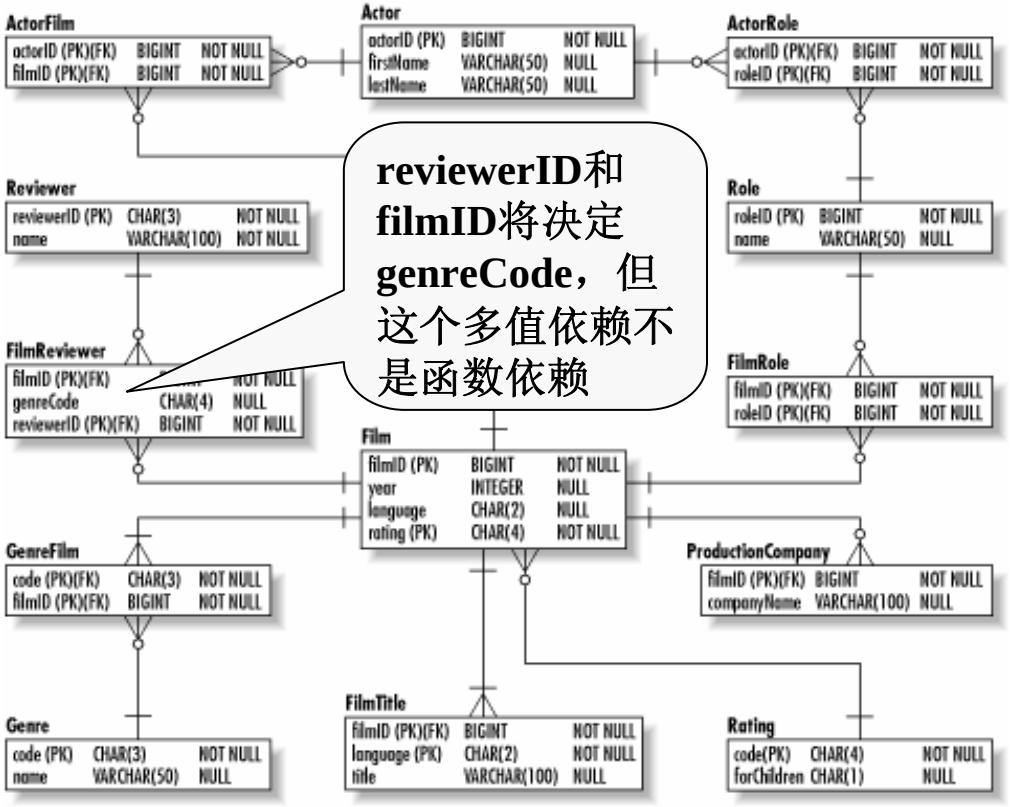
函数依赖：一个属性的取值被其它属性取值所决定

示例：符合2NF 但仍违反3NF

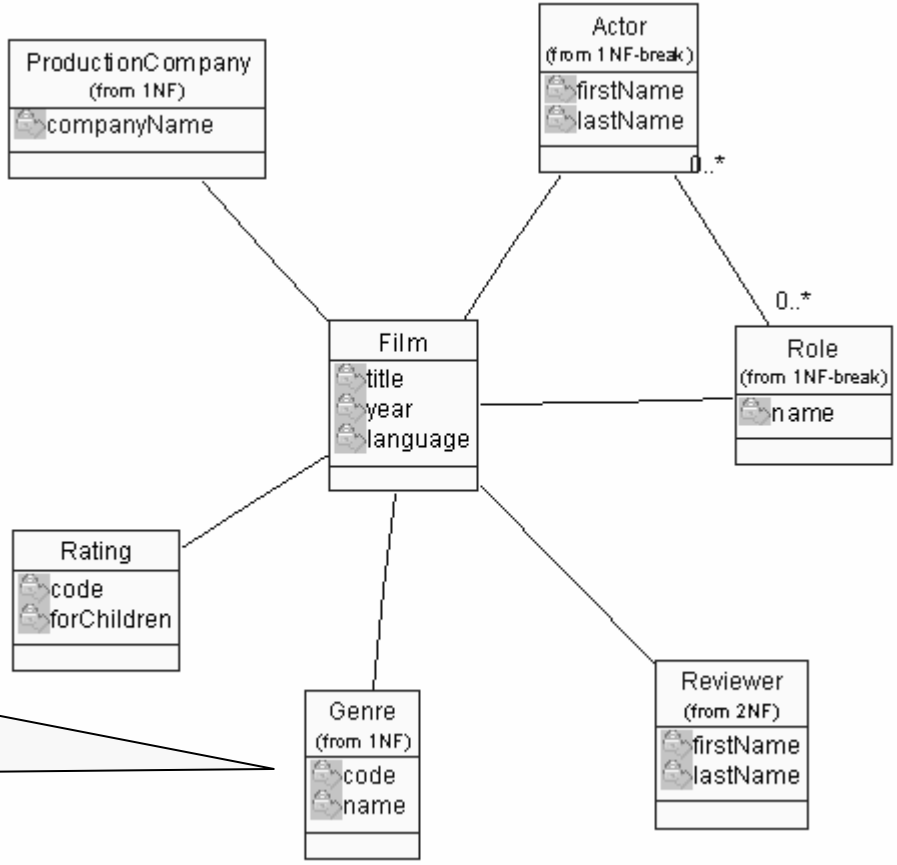


3NF:
实体的属性之间不存在转换**transitive**依赖

示例：符合3NF



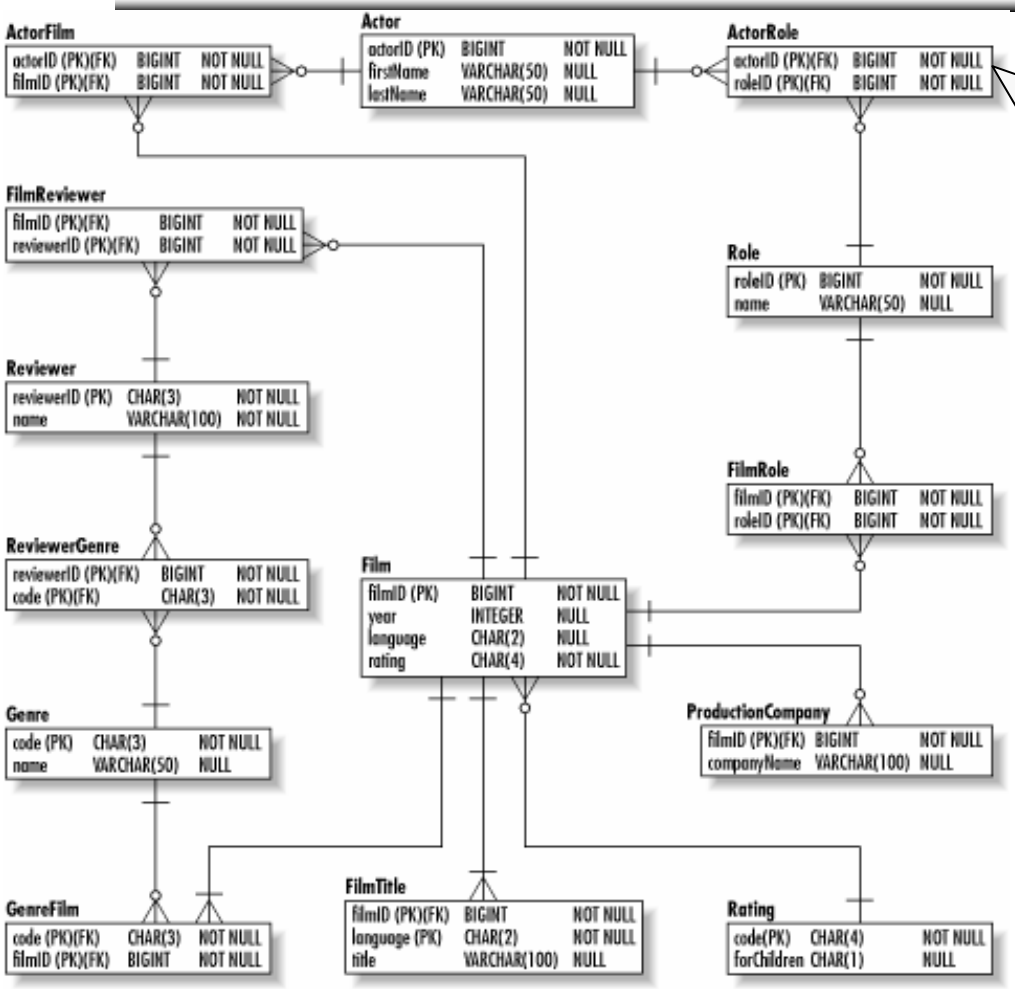
BCNF：
实体中所有决定性属性都是可选的键值，可以成为主键的一个组成部分



4NF：
满足**BCNF**，而且实体中所有多值依赖同时也是函数依赖

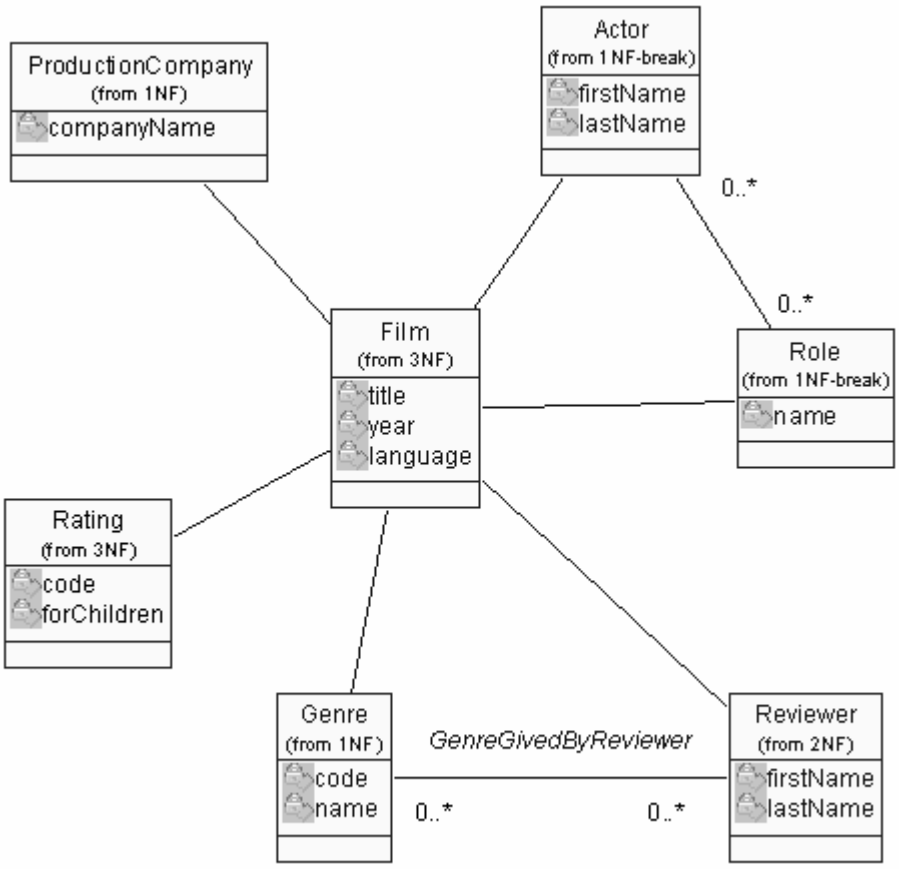
流派**Genre**由审核人**Reviewer**给定，因此存在一种新的关联

示例：符合4NF



演员Actor所扮演角色Role在此与影片Film无关，将造成在Join时出错

演员Actor应当在某影片Film中扮演一个角色Role



5NF：

当前的表结构不能无损地再向任意数量的更小表进行分解

示例：违反5NF

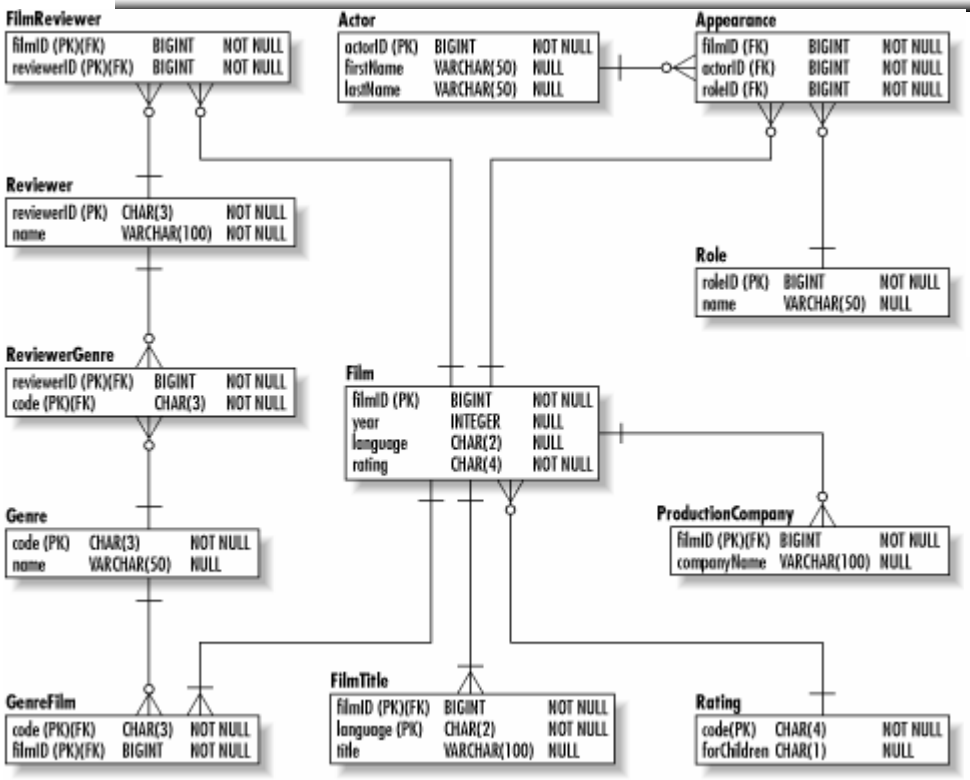
Joining actor, film, and role

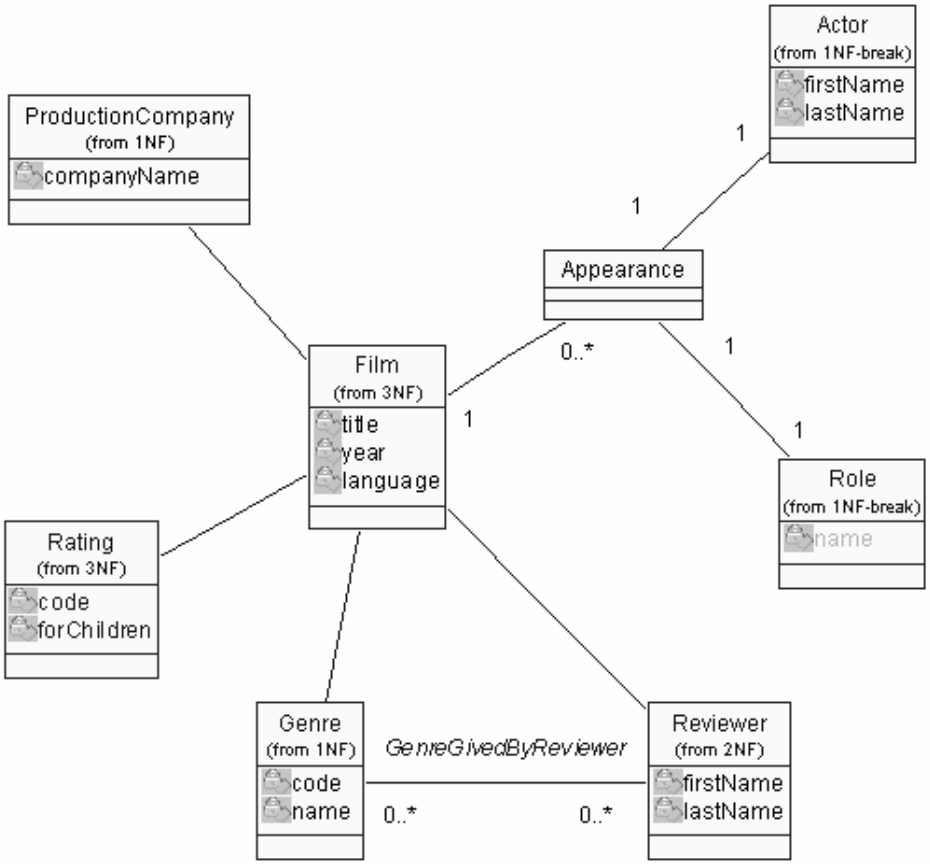
actorID	filmID	roleName	Description
1	101	The president	Stanley Anderson (1) played the president in Armageddon (101).
1	102	Edwin Sneller	Stanley Anderson (1) played Edwin Sneller in The Pelican Brief (102).

After adding Robert Culp who also played the president, but in the movie The Pelican Brief.

actorID	filmID	roleName	Description
1	101	The president	Stanley Anderson (1) played the president in Armageddon (101).
1	102	Edwin Sneller	Stanley Anderson (1) played Edwin Sneller in The Pelican Brief (102).
2	102	The president	Robert Culp (2) played the president in The Pelican Brief (102).
1	102	The president	Stanley Anderson (1) played the president in The Pelican Brief (102).
2	101	The president	Robert Culp (2) played the president in Armageddon (101).

示例：符合5NF


FilmReviewer
filmID (PK)(FK) BIGINT NOT NULL
reviewerID (PK)(FK) BIGINT NOT NULL
Actor
actorID (PK) BIGINT NOT NULL
firstName VARCHAR(50) NULL
lastName VARCHAR(50) NULL
Appearance
filmID (FK) BIGINT NOT NULL
actorID (FK) BIGINT NOT NULL
roleID (FK) BIGINT NOT NULL
Reviewer
reviewerID (PK) CHAR(3) NOT NULL
name VARCHAR(100) NOT NULL
Role
roleID (PK) BIGINT NOT NULL
name VARCHAR(50) NULL
ReviewerGenre
reviewerID (PK)(FK) BIGINT NOT NULL
code (PK)(FK) CHAR(3) NOT NULL
Film
filmID (PK) BIGINT NOT NULL
year INTEGER NULL
language CHAR(2) NULL
rating CHAR(4) NOT NULL
Genre
code (PK) CHAR(3) NOT NULL
name VARCHAR(50) NULL
ProductionCompany
filmID (PK)(FK) BIGINT NOT NULL
companyName VARCHAR(100) NULL
GenreFilm
code (PK)(FK) CHAR(3) NOT NULL
filmID (PK)(FK) BIGINT NOT NULL
FilmTitle
filmID (PK)(FK) BIGINT NOT NULL
language (PK) CHAR(2) NOT NULL
title VARCHAR(100) NULL
Rating
code(PK) CHAR(4) NOT NULL
forChildren CHAR(1) NULL


ProductionCompany
(from 1NF)
companyName
Actor
(from 1NF-break)
firstName
lastName
Appearance
Film
(from 3NF)
title
year
language
Role
(from 1NF-break)
name
Rating
(from 3NF)
code
forChildren
Genre
(from 1NF)
code
name
GenreGivedByReviewer
Reviewer
(from 2NF)
firstName
lastName
1
0..*

数据建模

面向对象的数据模型设计

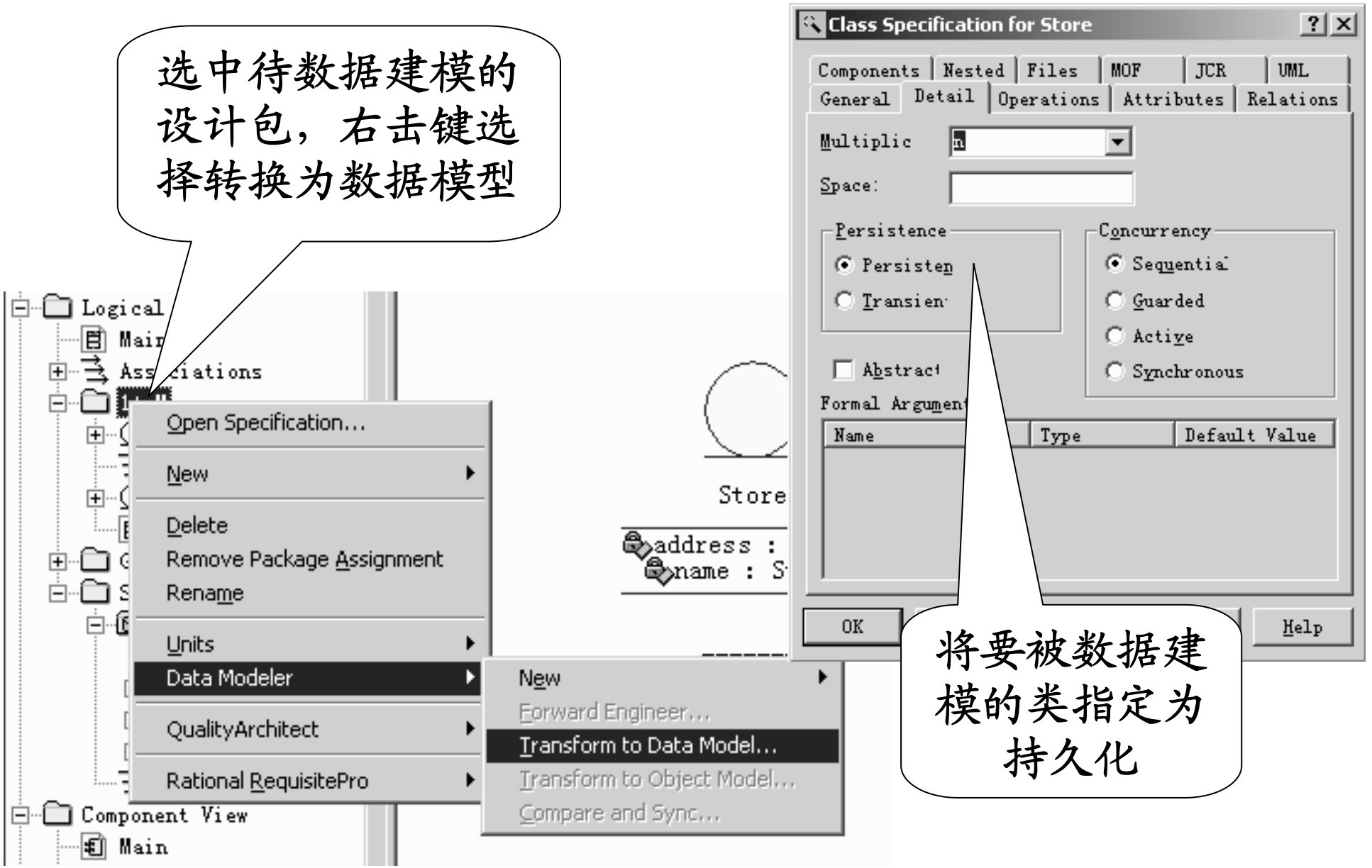
- 减少模型中实体和表的数量
- 增加系统结构的健壮性、灵活性和可维护性
- 用UML可以表示更多的相关业务规则
- 基本途径:
  - 将类映射为表，将对象映射为表的行，将对象的属性映射为表行中的列，将关系映射为外键

关系映射

- 一对一转换:
  - 为每个类分别创建一个表,每个表中的主键也是相关表中的外键
- 一对多转换:
  - 为每个类分别创建一个表,关联中“1”这一侧表的主键是“多”那一侧表的一个外键

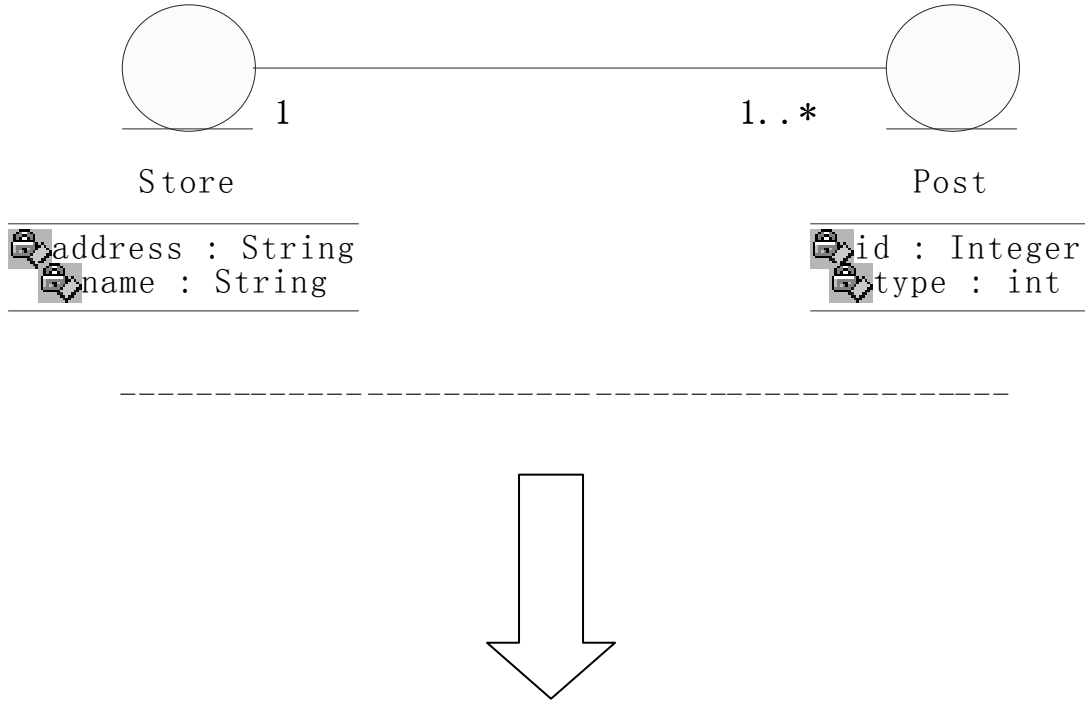
使用Rose的自动数据建模功能

选中待数据建模的设计包，右击键选择转换为数据模型



将要被数据建模的类指定为持久化

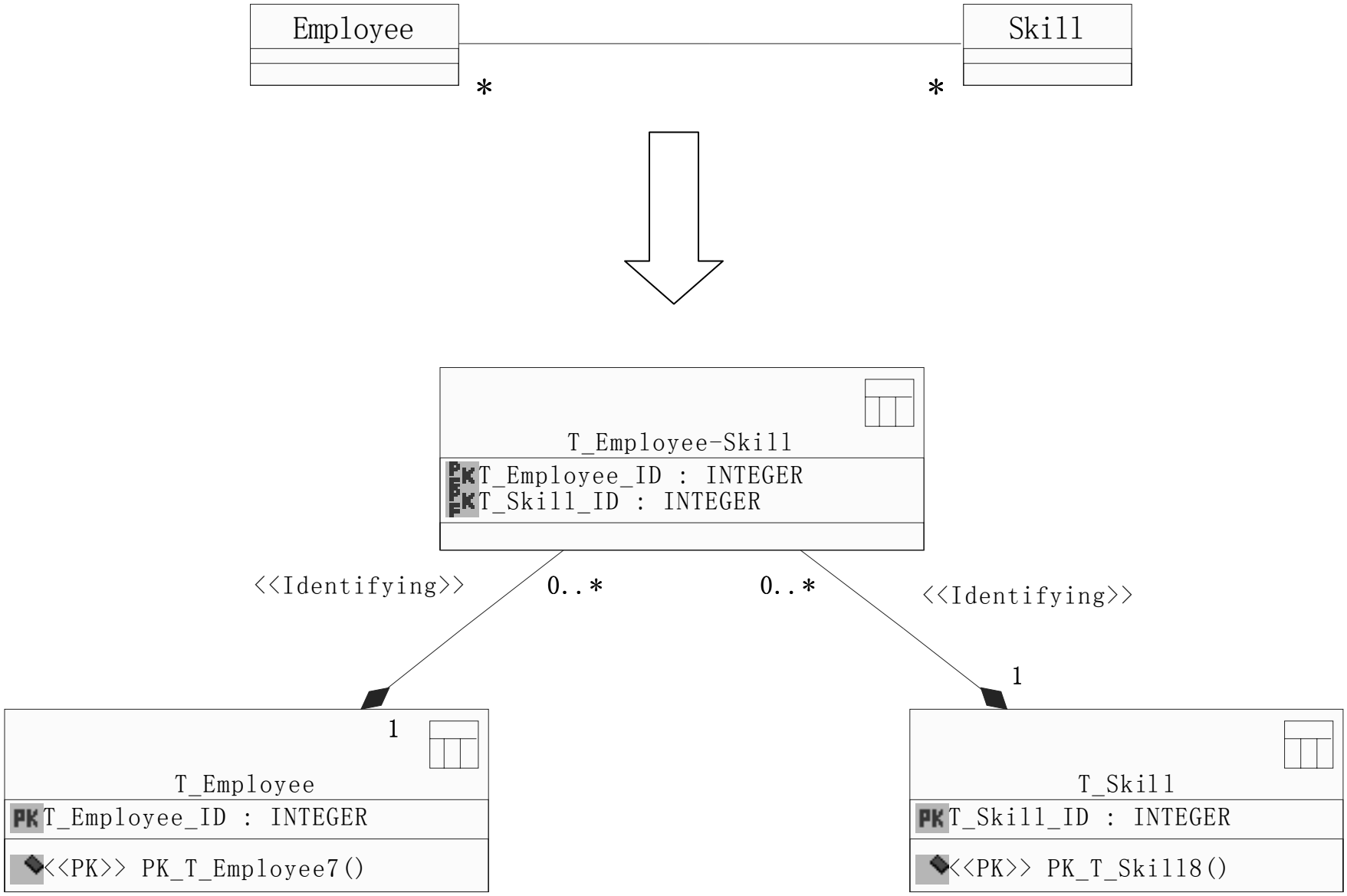
示例：一对多关系映射


1
1..*
Store
address : String
name : String
Post
id : Integer
type : int
<<Non-Identifying>>
T_Store
address : VARCHAR(255)
name : VARCHAR(255)
PK T_Store_ID : INTEGER
1
1..*
T_Post
PK id : INTEGER
type : SMALLINT
FK T_Store_ID : INTEGER

关系映射—多对多转换

- 为每个类分别创建一个表，每个表的主键在关联表中都定义为外键。
- 关联表的主键可以是单独的一列，也可能是两个外键组合再加上一个有含义的标识符
- 关联表映射

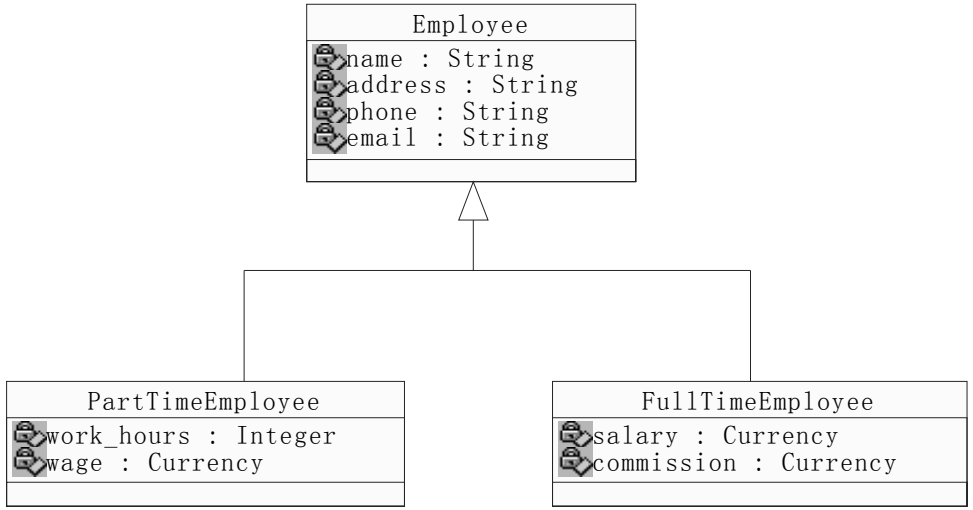
示例：多对多关系映射



关系映射—继承转换

- 为每个类分别创建一个表，为每个超类/子类对分别创建一个SQL视图（类表继承）
- 创建一个表（超类），并将子类的所有列信息加入到超类表中（上卷，单表继承）
- 对每个子类分别创建一个表，并将所有超类的列信息加入到每个子类的表中（下卷，具体表继承）

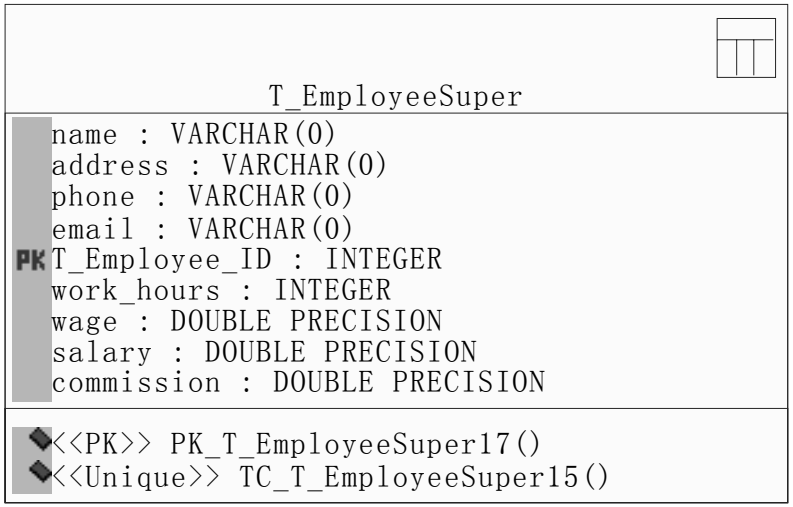
示例：继承关系映射



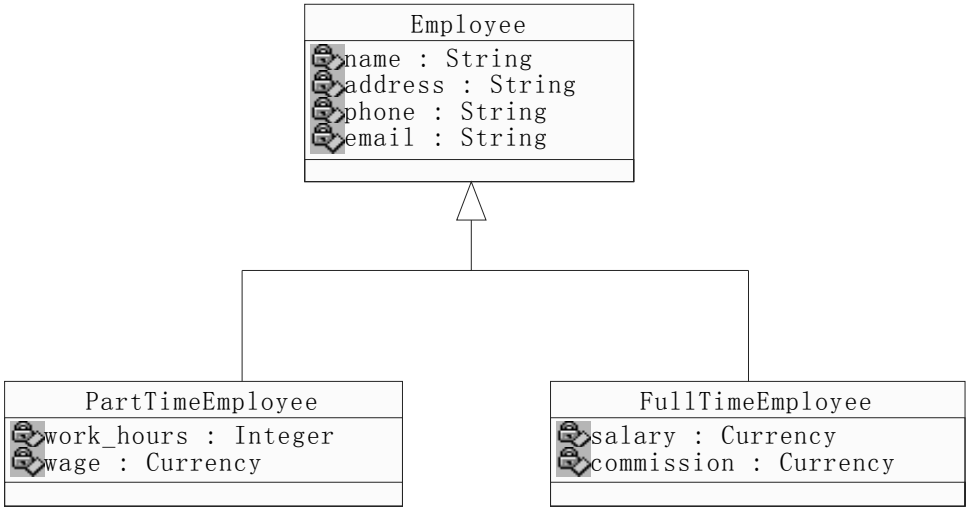
创建一个表（超类），并将子类的所有列信息加入到超类表中

Model only the superclass

If your queries rely heavily on data associated with the superclass and
If your queries do not rely on data associated with the subclasses and
If the subclasses do not contain a significant number of distinct attributes



示例：继承关系映射



Model only the subclasses

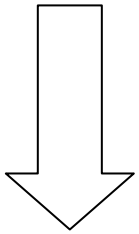
If your queries rely heavily on data associated with the subclasses and

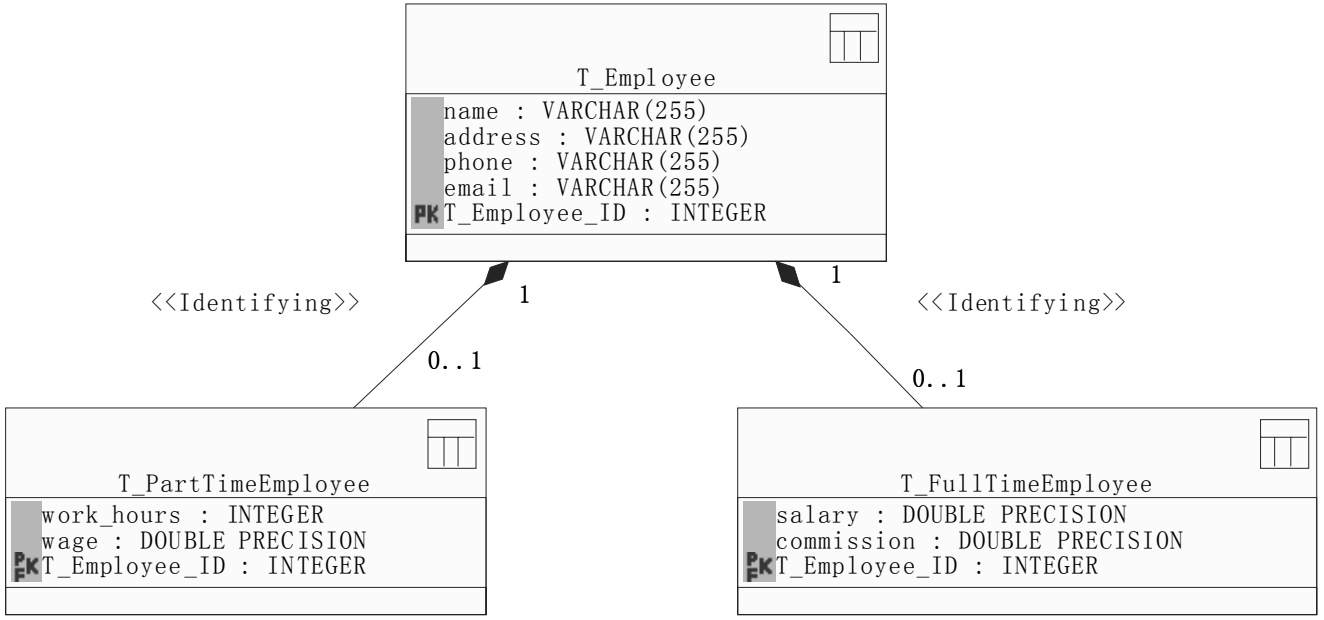
If your queries do not rely on data associated with the superclass

Model the superclass and its subclasses

For all other circumstances.

为每个类分别创建一个表

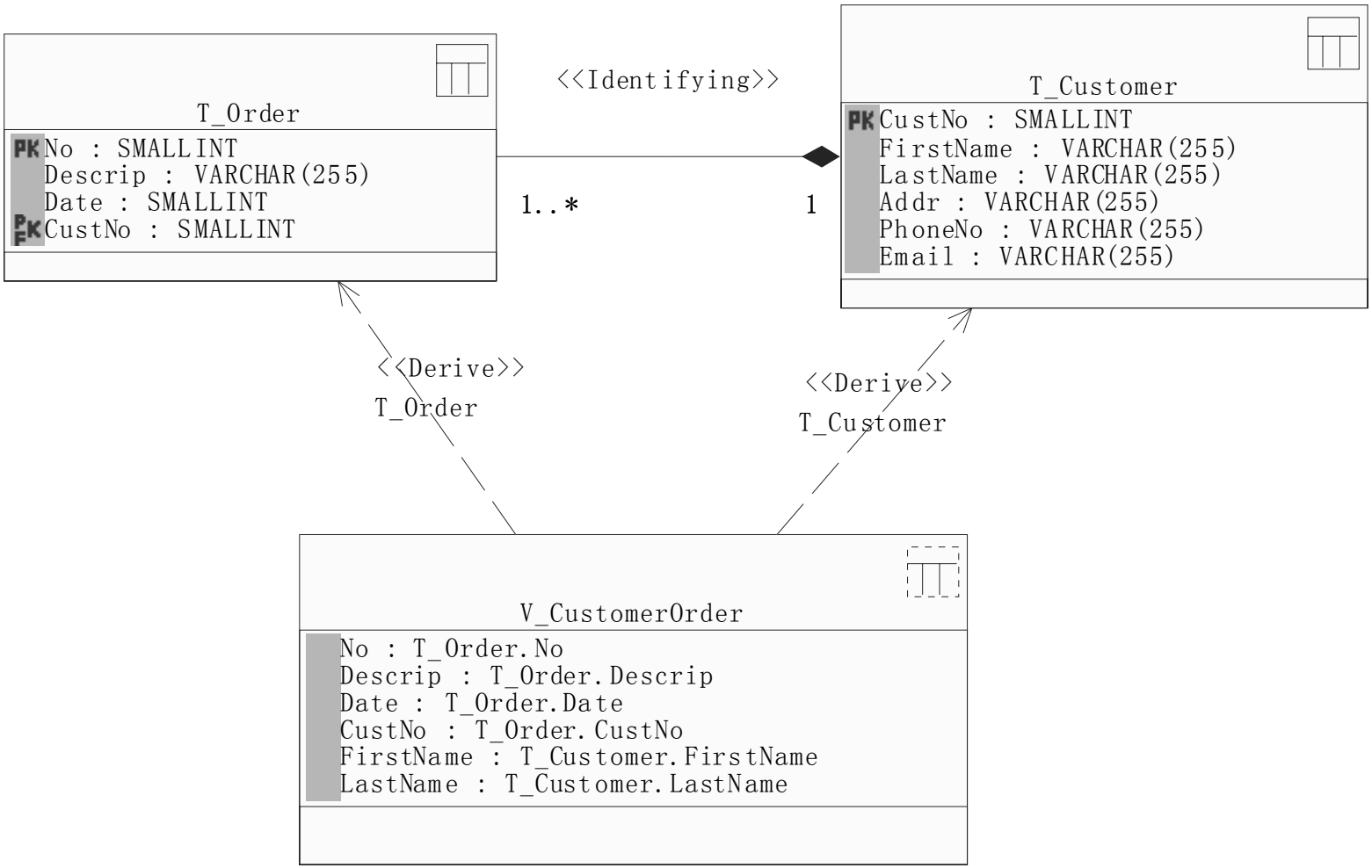




数据建模的元素

- 主键：用来唯一选择表的一行的候选键
- 外键：表中映射到另一张表的主键的一列或一组列
- 确定（identifying）关系：子表与父表共存
- 非确定（non- identifying）关系：每张表独立存在
- 视图：用户看到的虚表，具有典型的表的行为，不能独立存在
- 域：属性或列的一组有效赋值（用户自定义数据类型

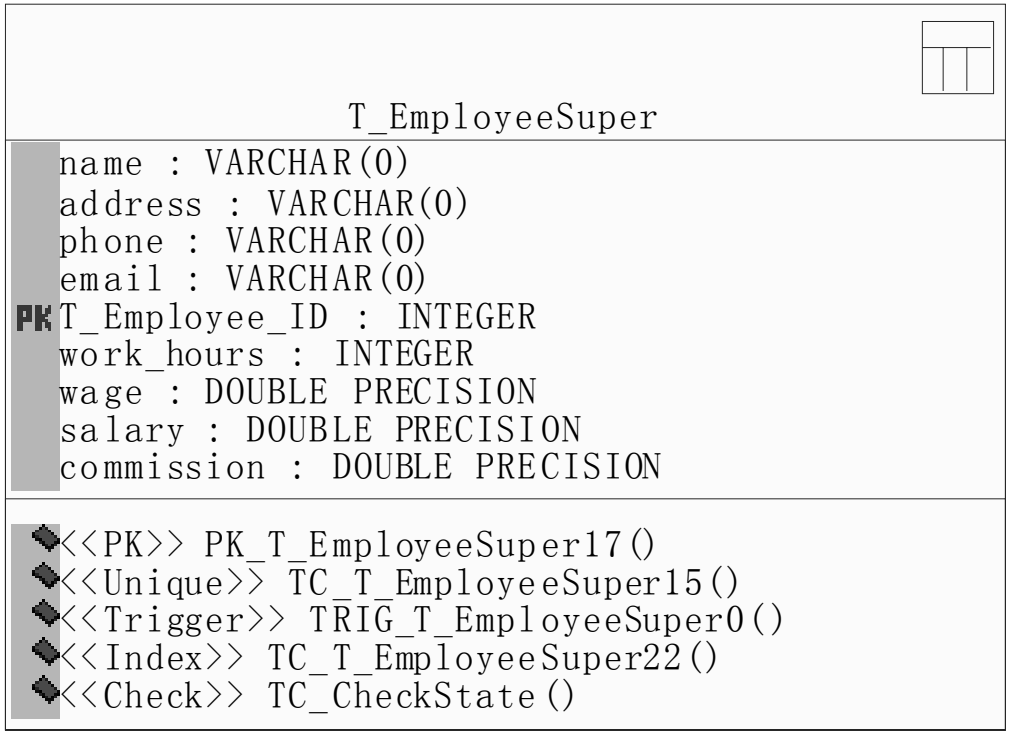
示例：视图


T_Order
PK No : SMALLINT
Descrip : VARCHAR(255)
Date : SMALLINT
PFK CustNo : SMALLINT
<<Identifying>>
1..*
1
T_Customer
PK CustNo : SMALLINT
FirstName : VARCHAR(255)
LastName : VARCHAR(255)
Addr : VARCHAR(255)
PhoneNo : VARCHAR(255)
Email : VARCHAR(255)
<<Derive>>
T_Order
<<Derive>>
T_Customer
V_CustomerOrder
No : T_Order.No
Descrip : T_Order.Descrip
Date : T_Order.Date
CustNo : T_Order.CustNo
FirstName : T_Customer.FirstName
LastName : T_Customer.LastName

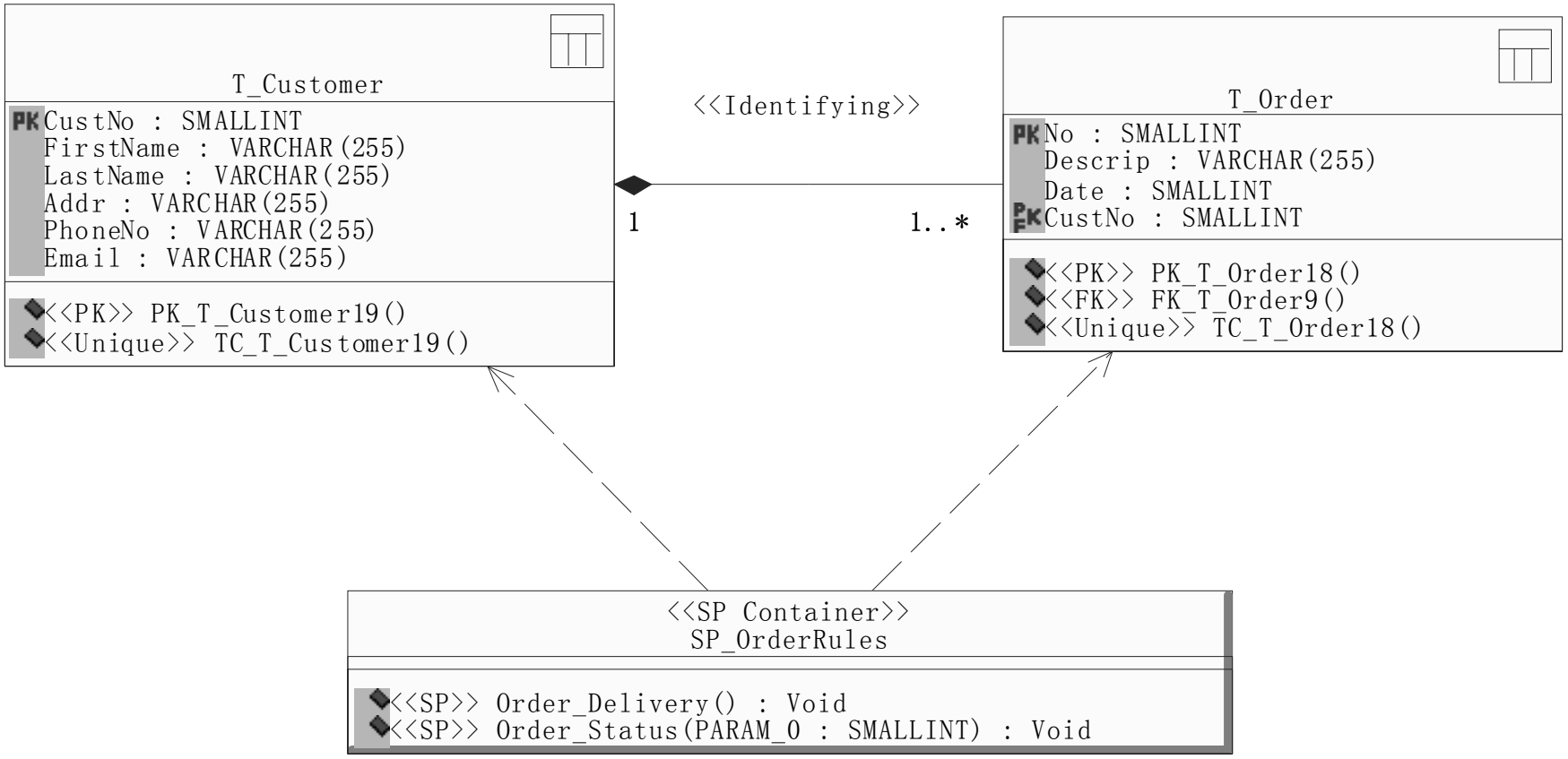
对列建模

- 键
- 约束
- 检查
- 唯一性
- 触发器
- 索引
- 空值
- 非空值
- 数据类型
- 长度
- 精度/量度

示例：触发器等项

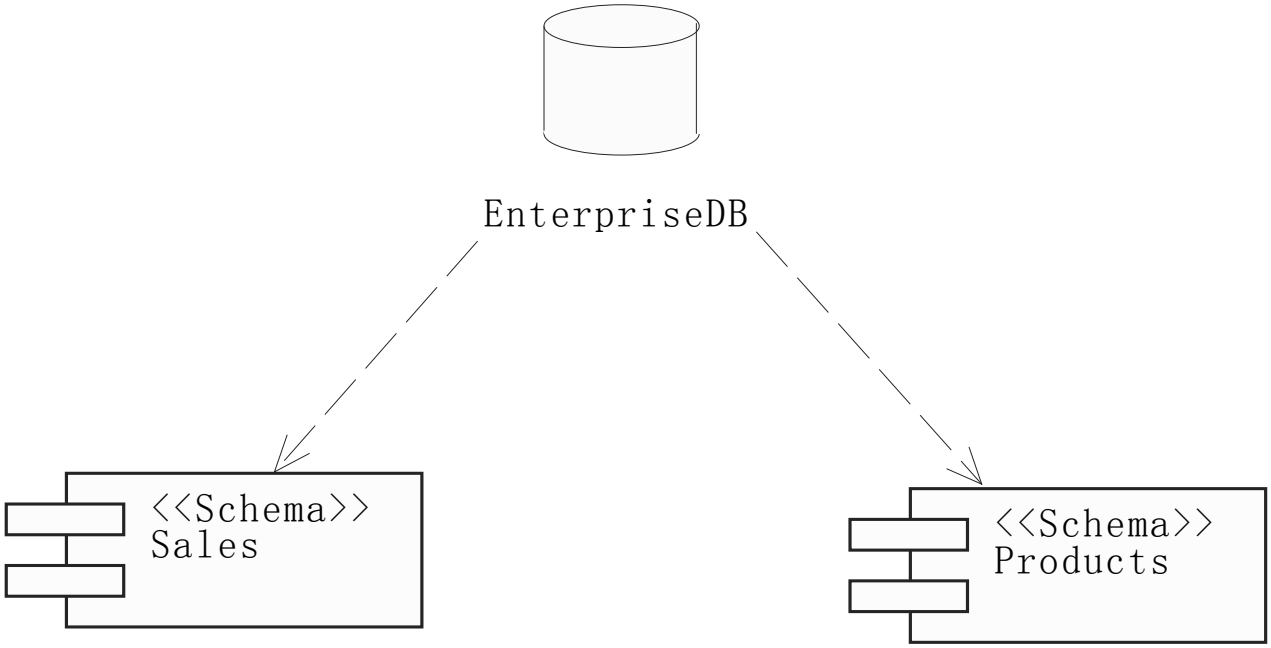

T_EmployeeSuper
name : VARCHAR(0)
address : VARCHAR(0)
phone : VARCHAR(0)
email : VARCHAR(0)
PK T_Employee_ID : INTEGER
work_hours : INTEGER
wage : DOUBLE PRECISION
salary : DOUBLE PRECISION
commission : DOUBLE PRECISION
<<PK>> PK_T_EmployeeSuper17()
<<Unique>> TC_T_EmployeeSuper15()
<<Trigger>> TRIG_T_EmployeeSuper0()
<<Index>> TC_T_EmployeeSuper22()
<<Check>> TC_CheckState()

示例：存储过程


T_Customer
PK CustNo : SMALLINT
FirstName : VARCHAR(255)
LastName : VARCHAR(255)
Addr : VARCHAR(255)
PhoneNo : VARCHAR(255)
Email : VARCHAR(255)
<<PK>> PK_T_Customer19()
<<Unique>> TC_T_Customer19()
<<Identifying>>
1
1..*
T_Order
PK No : SMALLINT
Descrip : VARCHAR(255)
Date : SMALLINT
PFK CustNo : SMALLINT
<<PK>> PK_T_Order18()
<<FK>> FK_T_Order9()
<<Unique>> TC_T_Order18()
<<SP Container>>
SP_OrderRules
<<SP>> Order_Delivery() : Void
<<SP>> Order_Status(PARAM_0 : SMALLINT) : Void

物理数据库

- 可把物理数据库看作Schema设计的具体实现;
- Schema提供了对永久信息的组织形式定义;
- 物理数据库模型表示了这些信息在数据库中的[illegible]



对象—关系映射的有关问题

- 唯一标识对象
- 类向表的映射
- 继承关系的映射
- 类间关联的映射
- 引入构架中的映射层
- 对象－关系映射的工具支持（TopLink）
- 需要处理各种可能的不同种类数据库错误
  - 死锁、内存耗尽、日志溢出、存储过程或sql查询语句的Bug、处理超时、服务崩溃、权限问题、并发重试、脏数据等

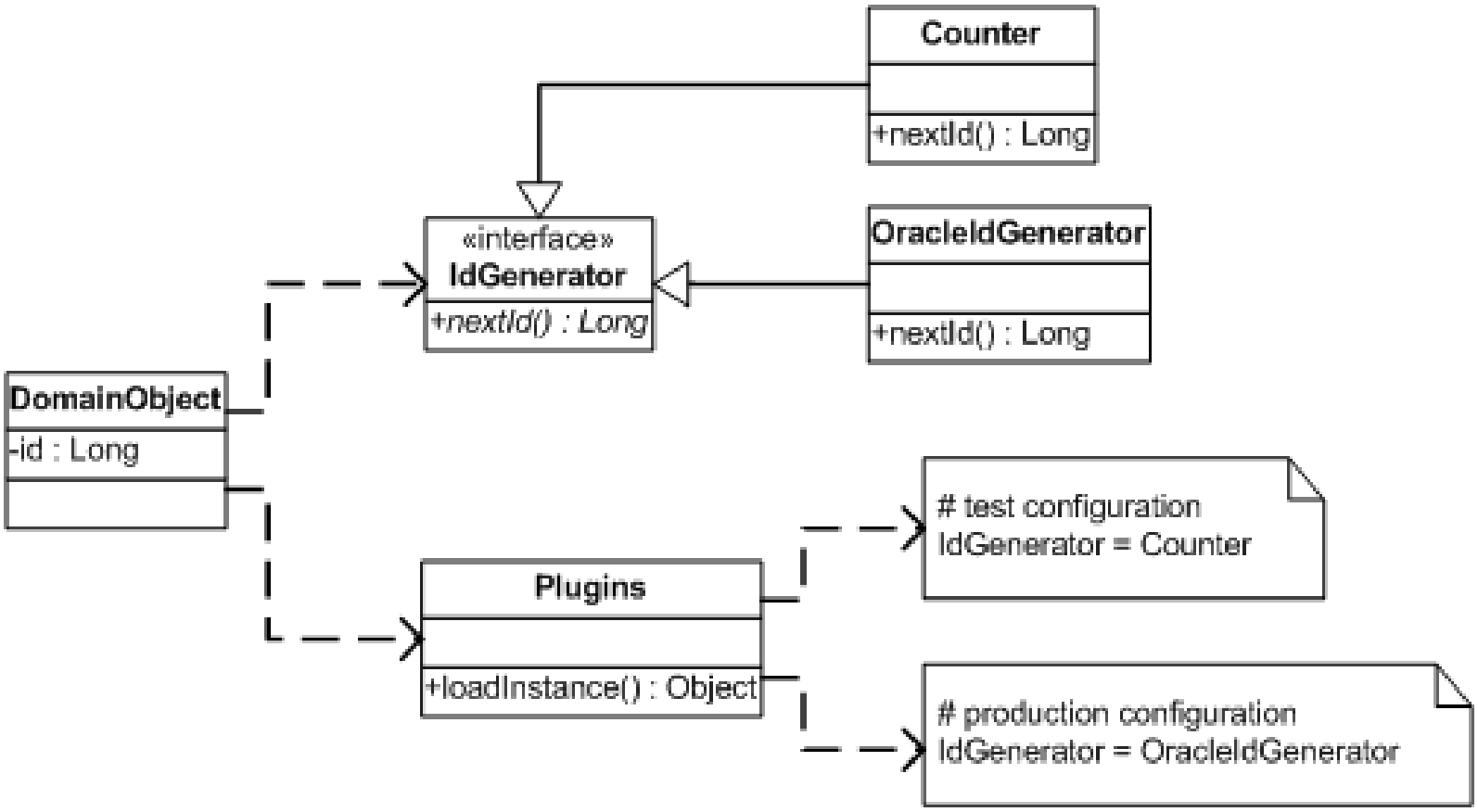
支持数据模型的模式

对象的唯一标识

- 每个对象都应当有一个全局唯一（在所有同一类和不同类的实例间）的标识来区分自己与其它对象；如果自己来生成和管理这个ID，可以使用64位的整形数,这样减少了回收ID的麻烦

- 优点：支持OO的概念，对象有与其属性无关的标识；将作为一种贯穿所有类的统一标识机制；DBMS对数值键处理效率高，有的还直接支持ID生成
- 缺点：许多类没有好的天然主键（例如人、地址）；用户可能想要基于领域属性来访问数据；造成间接的查找，效率为O(n)；有些RDBMS不支持对象的视角

实例：对象标识生成器

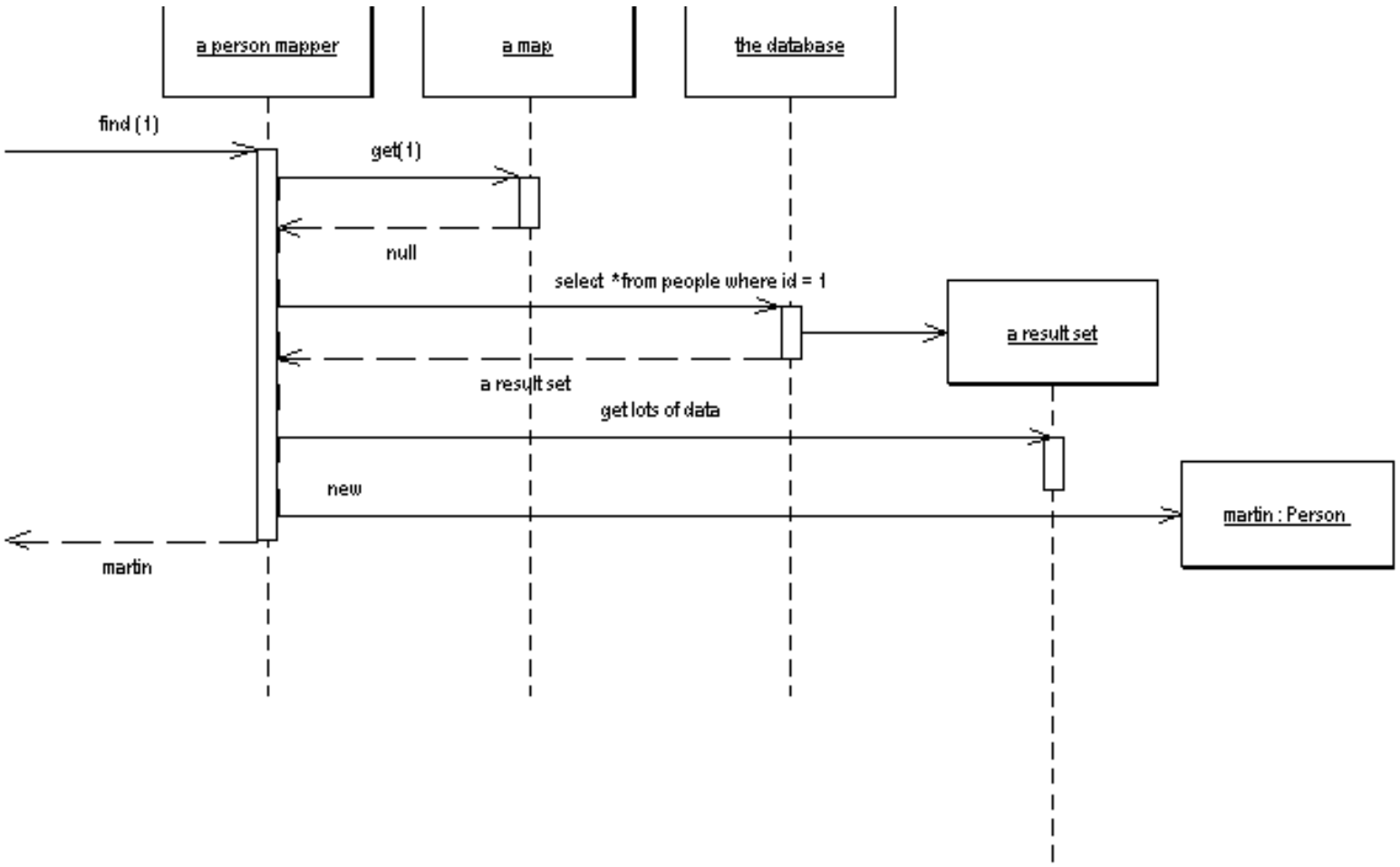

Counter
+nextId() : Long
«interface»
IdGenerator
+nextId() : Long
OracleIdGenerator
+nextId() : Long
DomainObject
-id : Long
Plugins
+loadInstance() : Object
# test configuration
IdGenerator = Counter
# production configuration
IdGenerator = OracleIdGenerator

数据映射模式

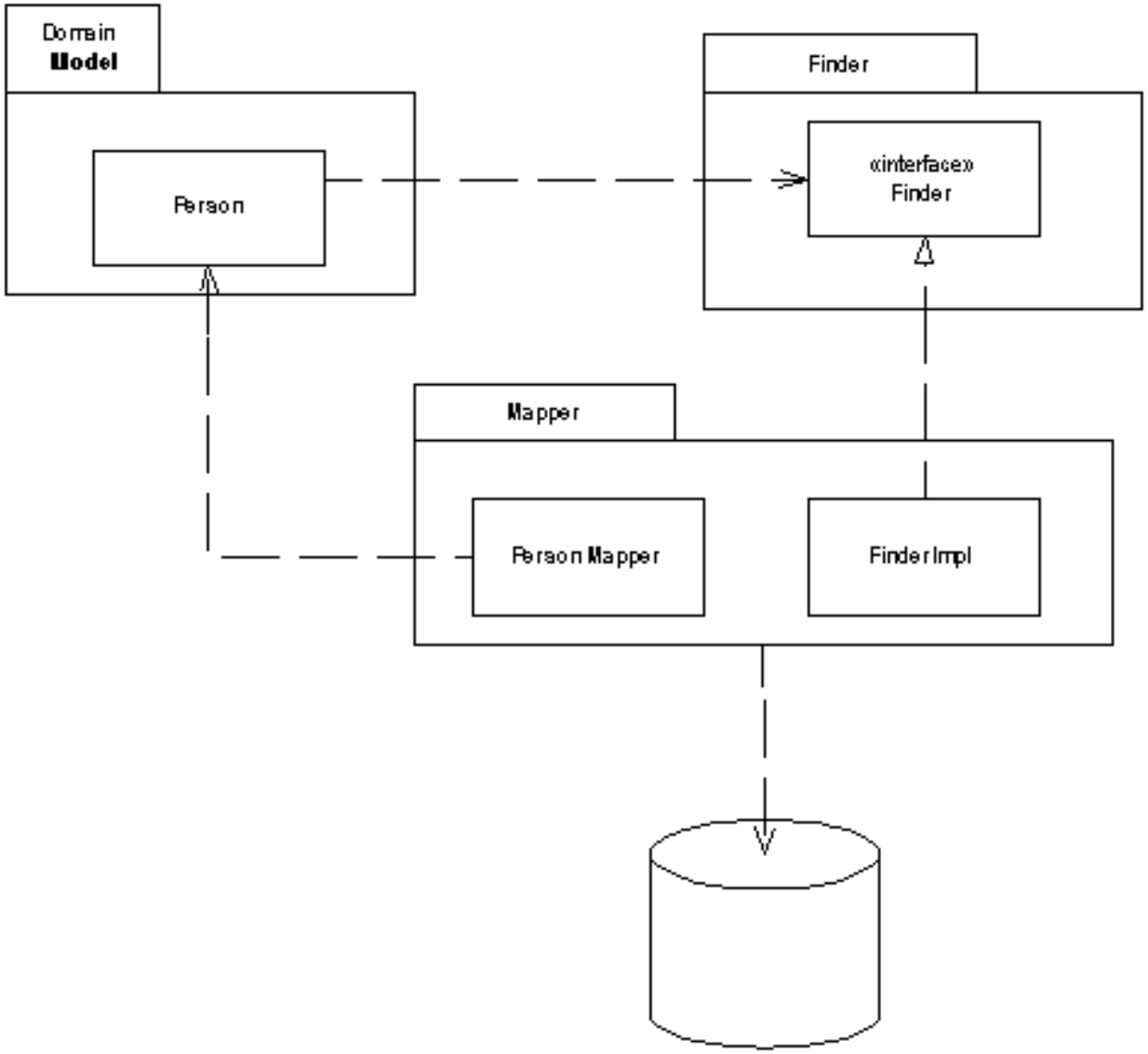
- 在面向对象应用中使用关系数据库，首要的就是如何解决对象模型和关系模型的不匹配问题
- 数据建模中已经将类映射为表，那么如何在代码中将对象写到对应表中，和从表中读出对象的数据？
- 我们可以为持久对象分别定义专门的类来处理这些繁杂的工作；由数据映射类来掌握持久类与表的对应关系，以及类的关联与外键的关系等

实例：数据映射模式


Person
getExemptions
Person Mapper
insert
update


a person mapper
a map
the database
find (1)
get(1)
null
select *from people where id = 1
a result set
a result set
get lots of data
new
martin : Person
martin

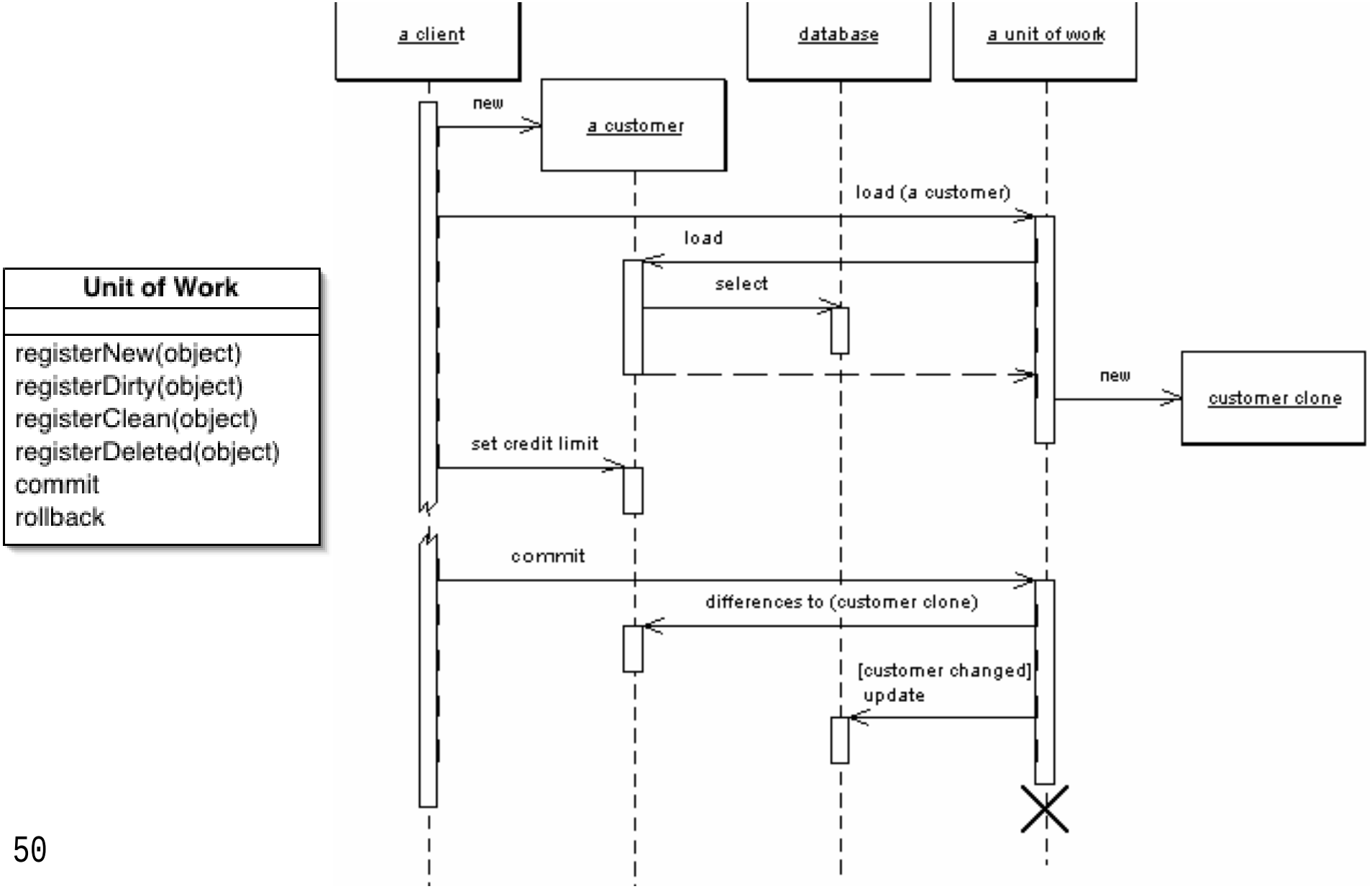
实例：数据映射查找器


Domain Model
Person
Finder
«interface»
Finder
Mapper
Person Mapper
Finder Impl

UnitOfWork模式

- 当执行对数据库的读写操作时，使用UnitOfWork对象来记录所有针对数据的变动，并延迟对数据库的回写操作，从而避免了对数据库的频繁调用
- 在执行回写时，只有那些在UnitOfWork对象中记录的变化部分才需要真正回写，从而极大地减少了对数据库的存取开销
- 本模式的限制在于要求在提交UnitOfWork之前的所有交互都属于同一次事务，并要求事务在此间一直保持开放

实例： UnitOfWork模式



领域逻辑模式

- 事务脚本
- 领域模型
- 表模块

数据源架构模式

- 表数据关口
- 行记录关口
- 主动记录

参考资料

<UML Distilled: A Brief Guide to the Standard Object Modeling Language>
<UML Reference Manual>
<Mythical Man Month>