

《交通大数据技术》



2024/6/14

交通大数据技术

马晓磊
交通科学与工程学院
2024年春季

关系模型



实例vs.模式

实例:

- 数据库中包含的特定数据

数据库模式与结果数据库的实例完全不同。

- 模式随着时间的推移会保持稳定—属性几乎永远不会改变。
- 数据库实例随着数据的添加，删除和更新而不断变化。
- 对于数据库设计，尽管典型实例的知识有助于指导我们的设计，但只有模式是最重要的。

从E/R图到关系模式

创建一个新数据库的过程

- 设计阶段-考虑我们需要存储的信息、关系和约束
- 实现阶段-在真实的DBMS中构建数据库

大多数商业DBMS使用关系模型，我们也需要使我们的设计与此模型保持一致。

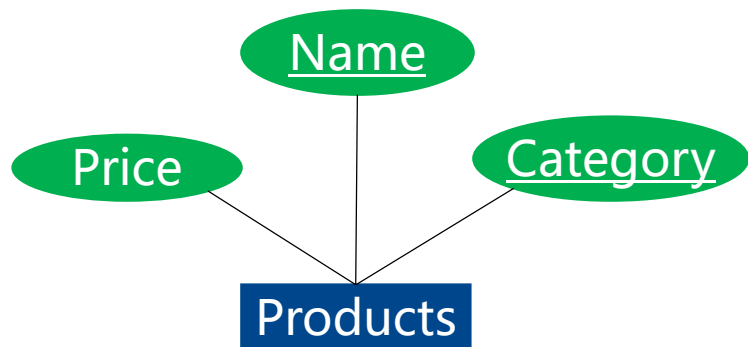
这涉及将E / R图设计转换为关系数据库，非常简单。

- 将每个实体集转换为具有相同属性集的关系。
- 将关系替换为属性是关联实体集的键的关系。

从实体集到关系

首先考虑强实体集。

创建具有相同名称和相同属性集的关系。



Products(Name, Category, Price)

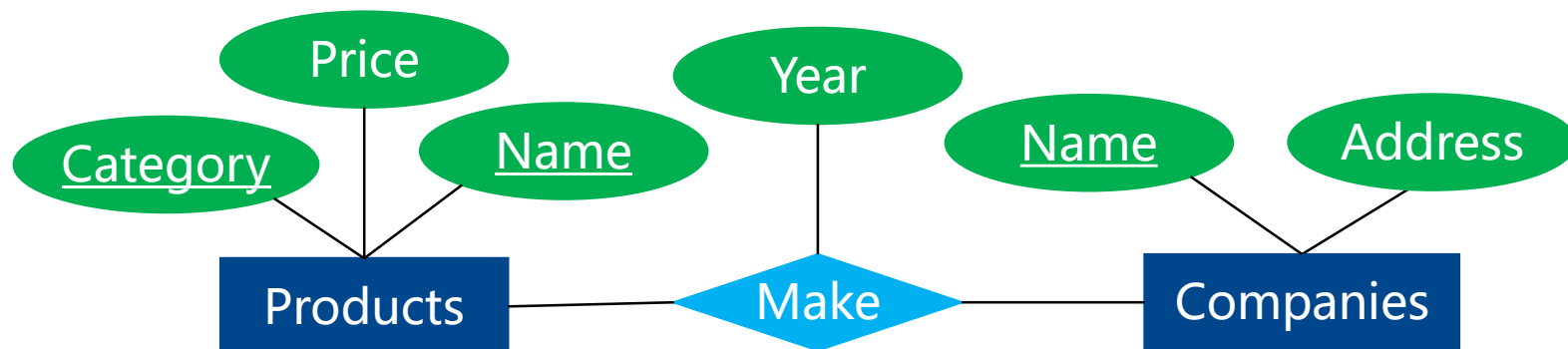
Name	Category	Price
VX720	Monitor	\$199.99

从“表间关系”到“关系表”

当 E 到 F 有一个多对多关系 R 时，表间关系 R 可以转换成具有如下属性的联系：

- E 的关键属性
- F 的关键属性
- 属于关系 R 的任何属性

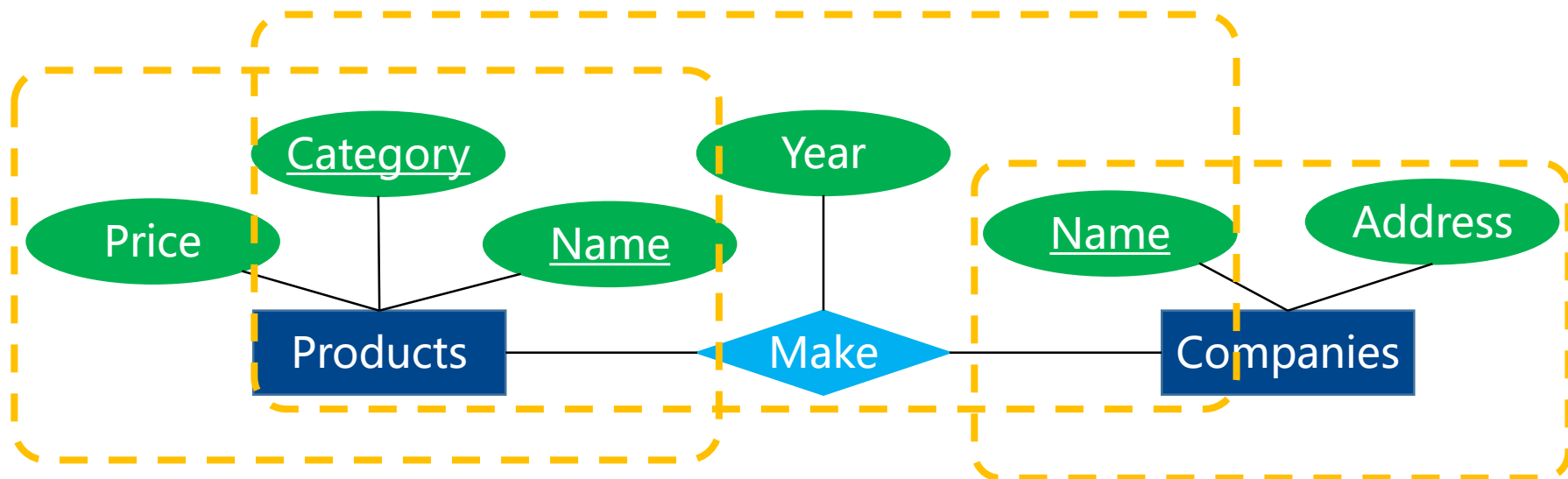
从“表间关系”到“关系表”



Make(Products.Name, Products.Category, Companies.Name, Year)

Products.Name	Products.Category	Companies.Name	Year
VX720	Monitor	Gateway	1999

从“表间关系”到“关系表”



Companies(Name, Address)

Products(Name, Category, Price)

Make(Products.Name, Products.Category, Companies.Name, Year)

合并关系表 (Relations)

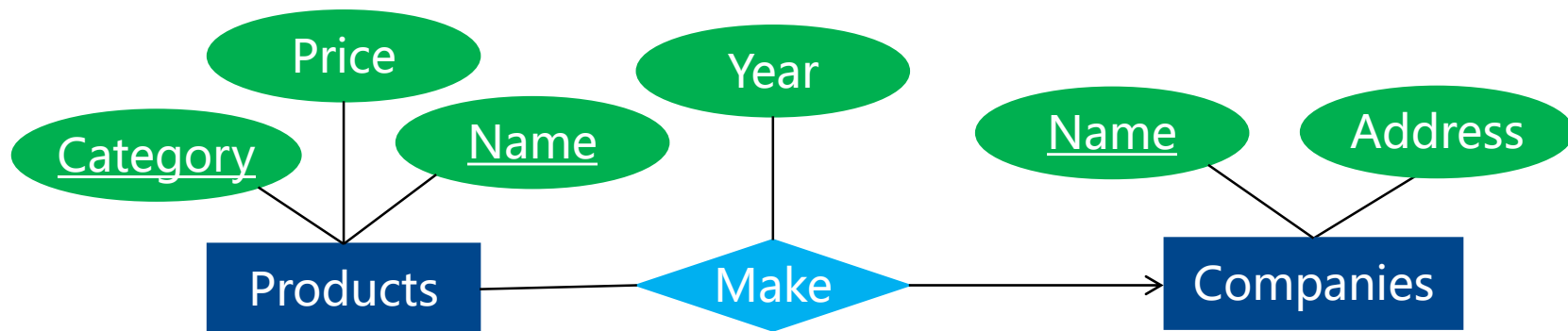
如何处理多对一关系?



当E到F有一个多对一关系时，R和E需要通过一个包含以下内容的模式组合为一个关系：

- E的所有属性
- F的主属性
- 属于R的任何属性

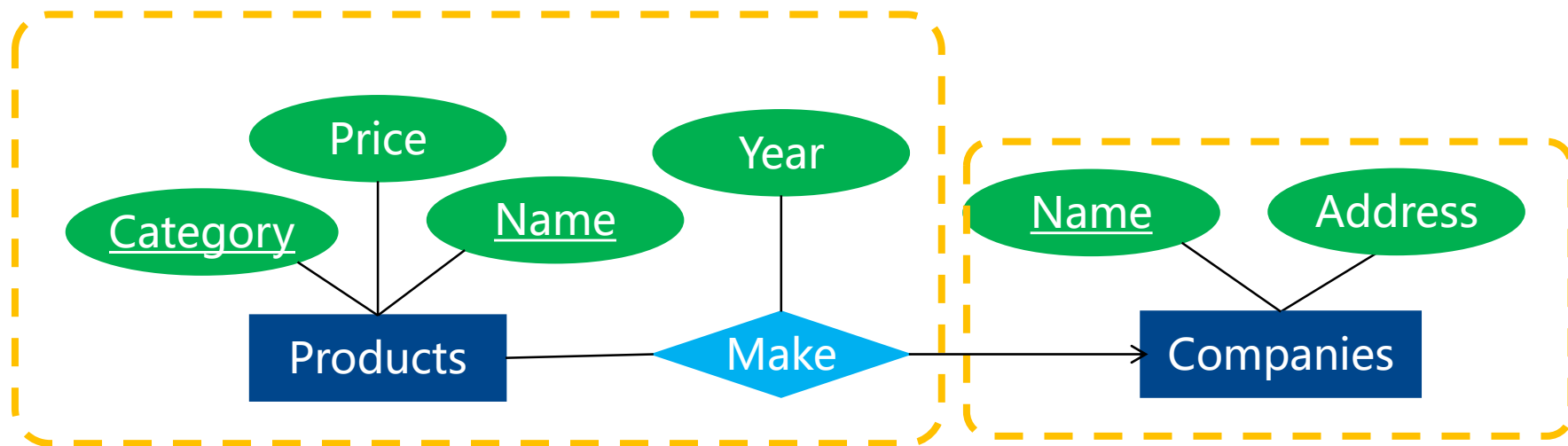
合并关系表



我们需要多少关系表?

两个! 不需要Make. Products, 可以修改为包括Year和Companies.Name的Products表, Companies是另外一个关系表。

合并关系表



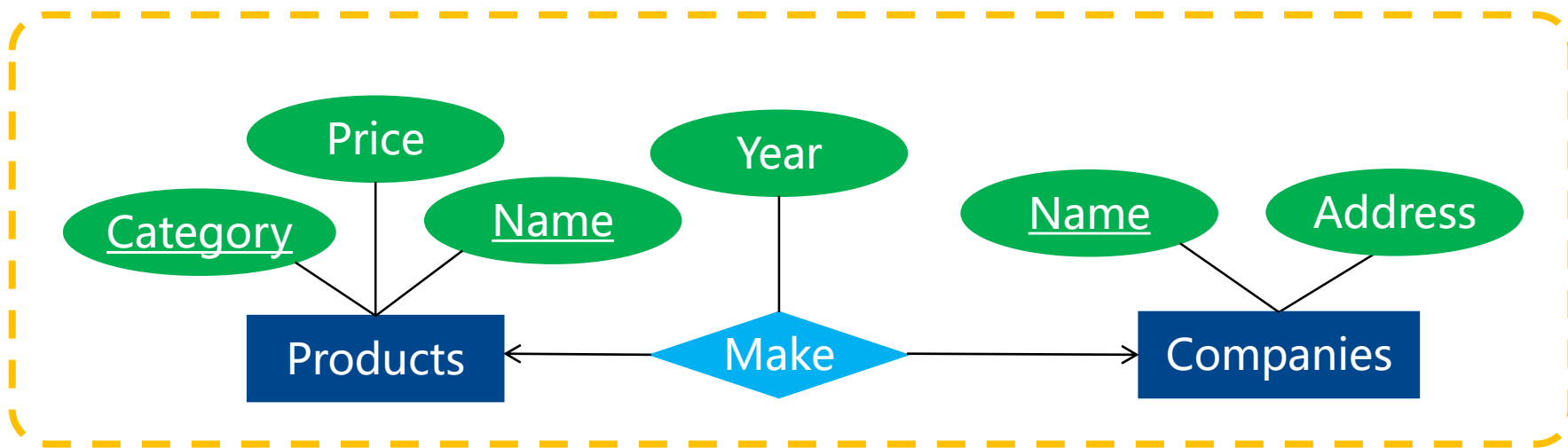
Companies(Name, Address)

Products(Name, Category, Price, Companies.Name, Make.Year)

合并关系表

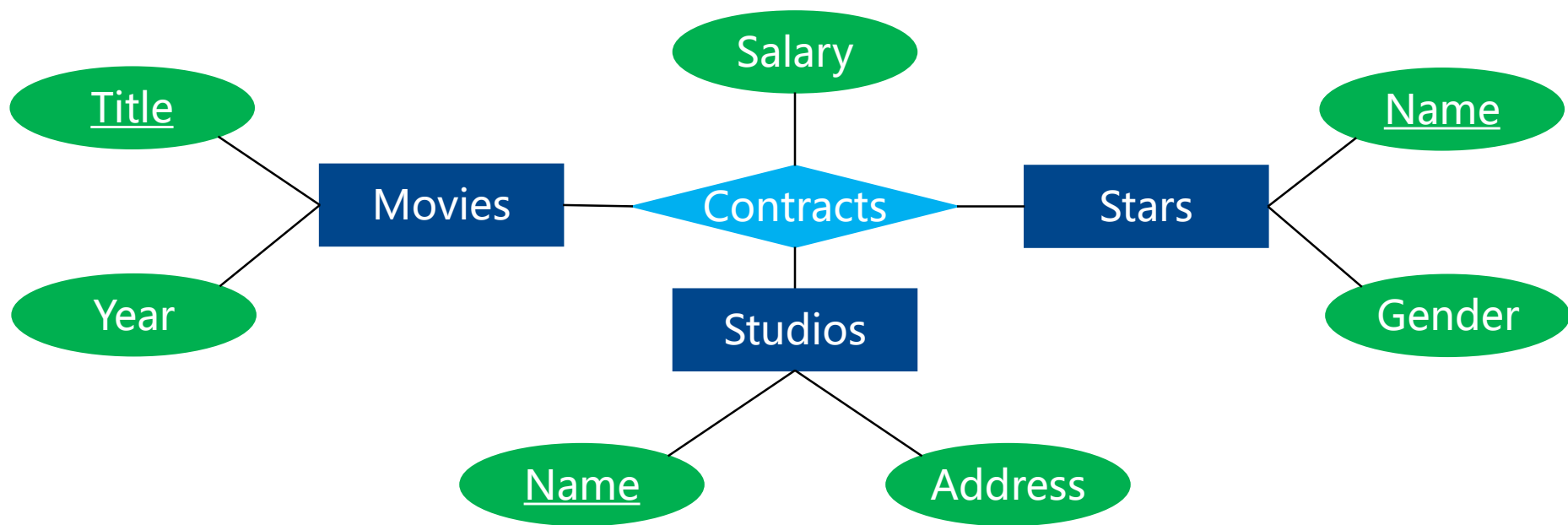
一对一关系

- 两个实体集的整个关系变成了一个包含所有属性的大关系
- 主键可以从我们选择作为主实体的任何一个实体中选取



CompaniesAndProducts(Companies.Name, Companies.Address, Products.Name, Products.Category, Products.Price, Make.Year)

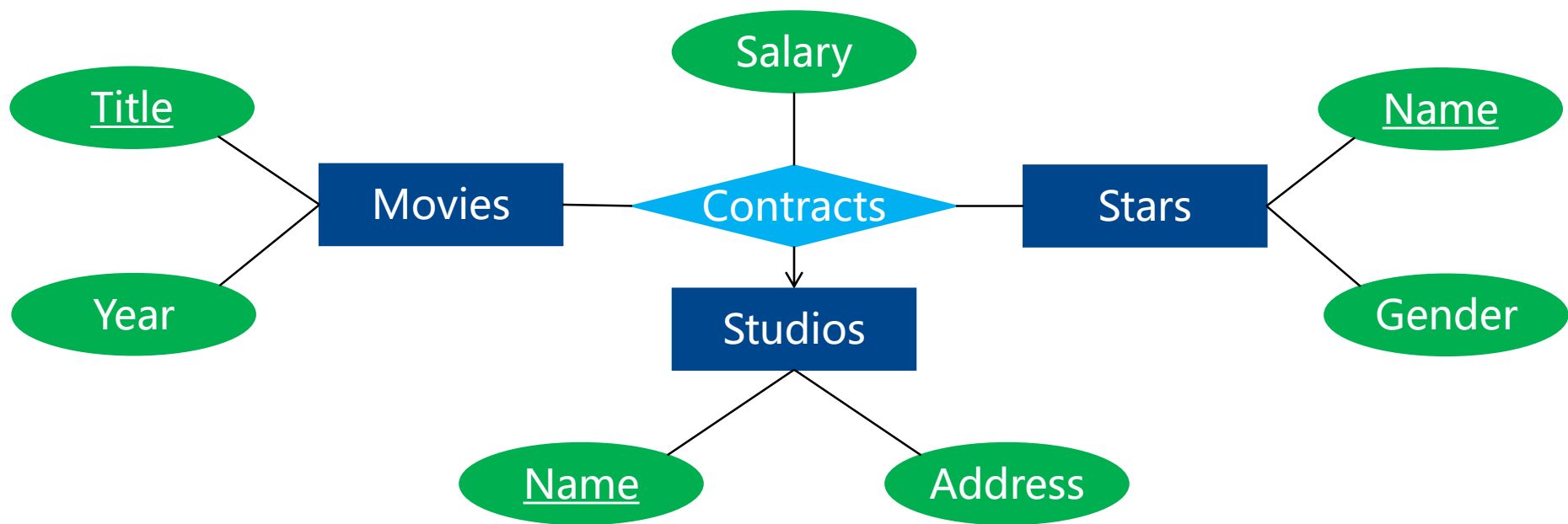
从“表间关系”到“关系表”



所有实体集都转换为关系表，表间关系Contracts也是如此：

`Contracts(Movies.Title, Stars.Name, Studios.Name, Salary)`

从“表间关系”到“关系表”

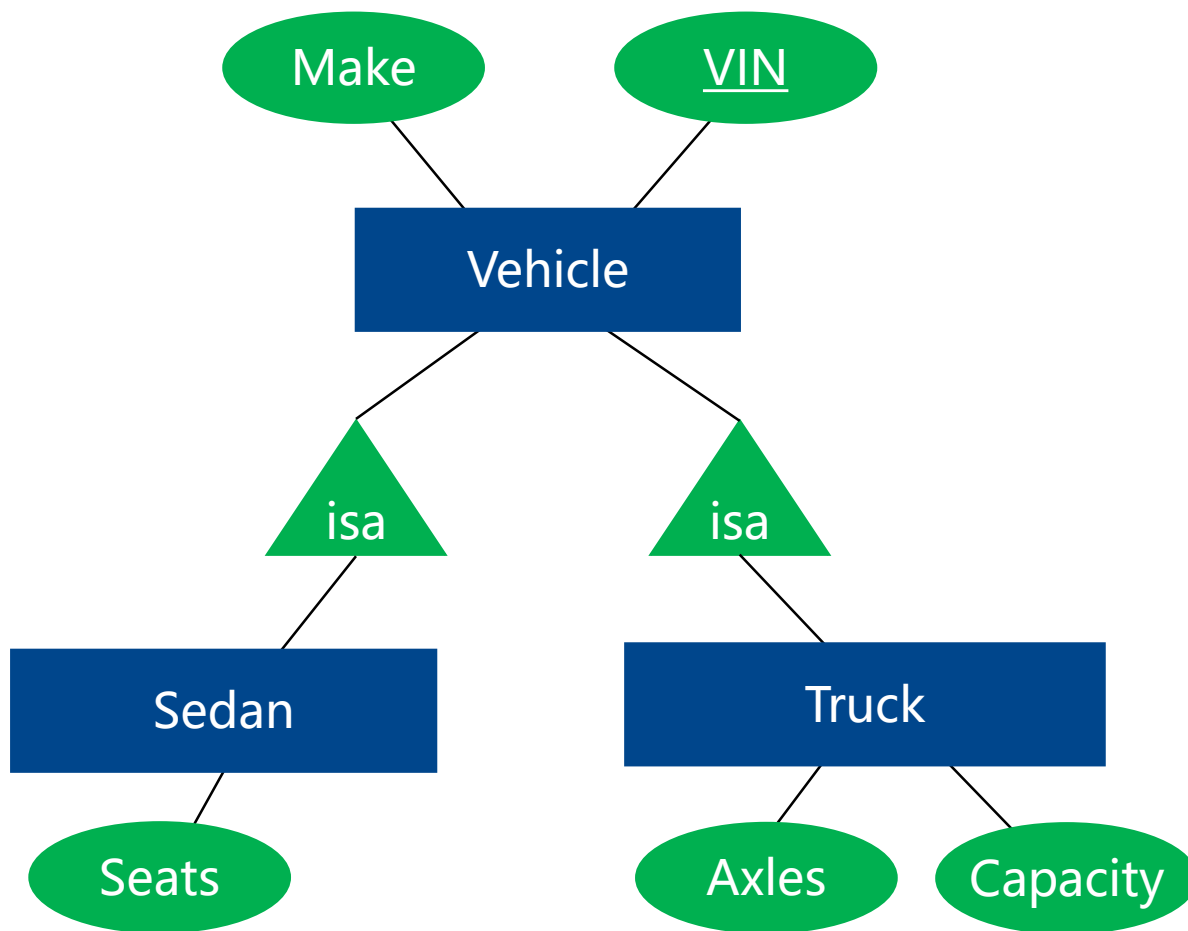


三者之间的关系，两边是多对一，一边是一对一。

`Contracts(Movies.Title, Stars.Name, Studios.Name, Salary)`

与之前的幻灯片有什么不同？

从子类到关系表



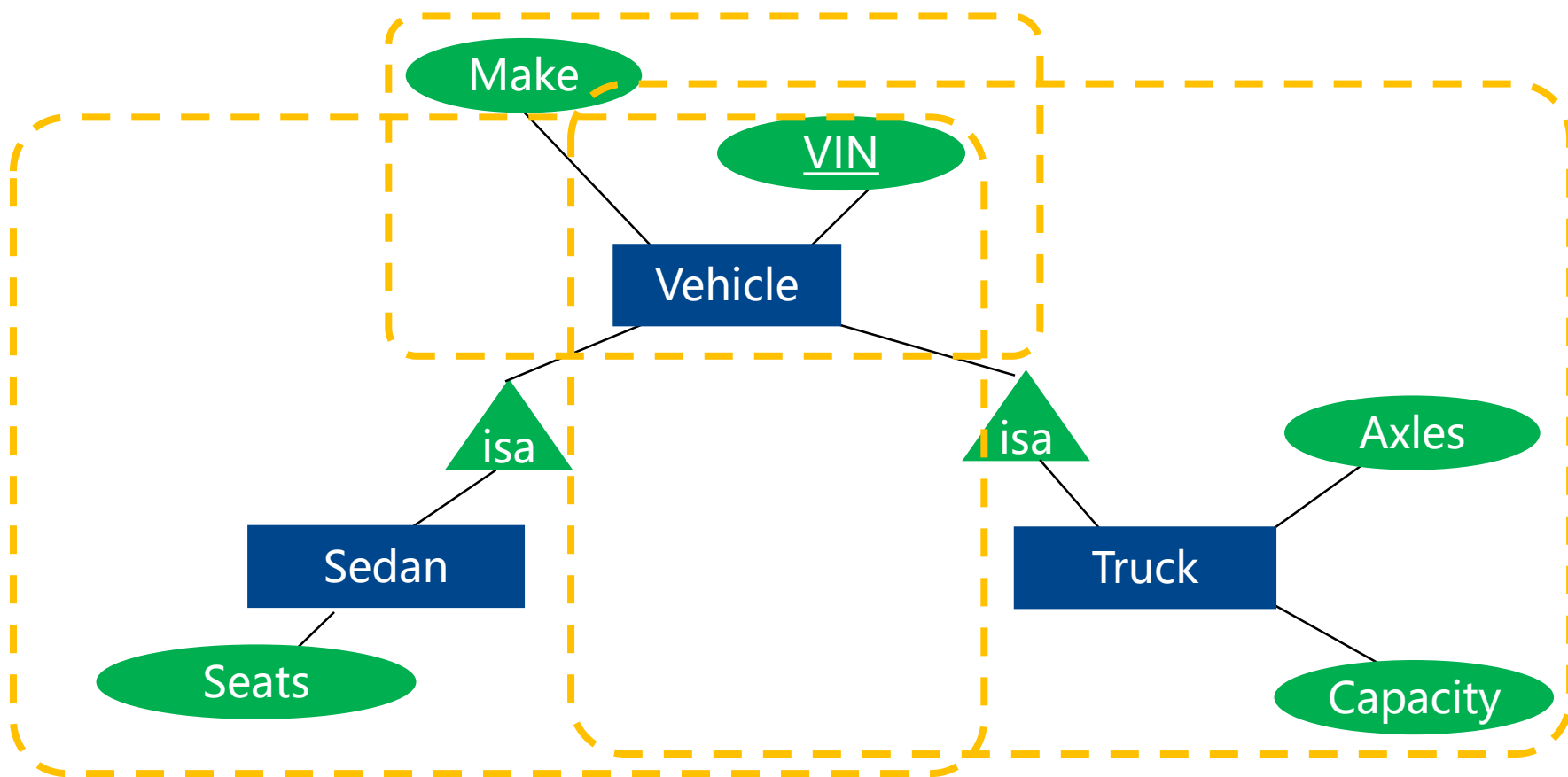
从子类到关系表

有三种方法可以将子类实体转换为关系表：

- 面向对象——每个实体集都成为一个关系表，具有所有适用的属性
- 空值——用一个表来表示整个层次结构，空值表示不适用的属性
- ER 方法——我们将使用的方法

对于层次结构中的每一个实体集E，创建一个包含来自根的主属性和所有属于E的属性的关系表

从子类到关系表

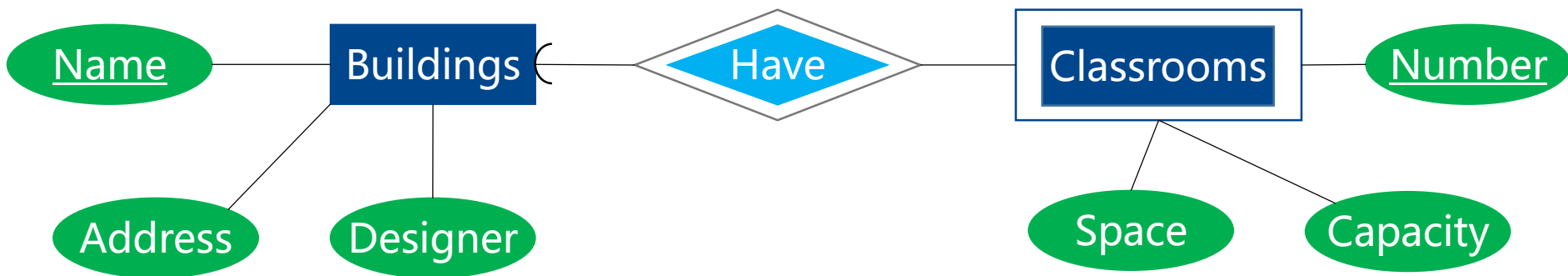


Vehicle(VIN, Make)

Sedan(Vehicle.VIN, Seats)

Truck(Vehicle.VIN, Axles, Capacity)

处理弱实体集



对于弱实体集，支持关系必须是具有引用完整性的多对一关系。

与多对一关系类似，我们将关系和弱实体集结合起来创建一个关系表。

Buildings(Name, Address, Designer)

Classrooms(Buildings.Name, Number, Space, Capacity)

关系模型设计

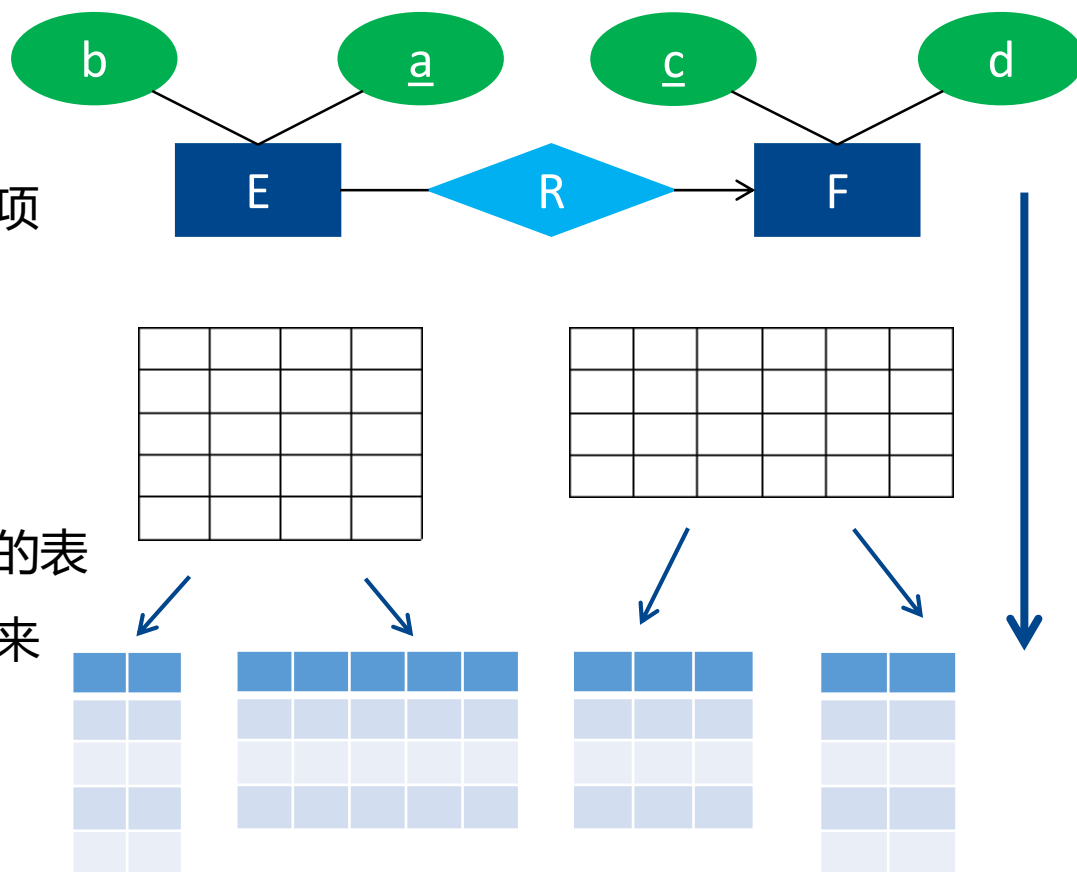
概念模型：

关系型模型

- 可能包含不需要的函数依赖项

规范化：

- 消除异常
- 减少冗余和依赖
 - 将较大的表划分为较小的表
 - 使用关系将它们链接起来



函数依赖

关系表R上的**函数依赖(FD)** 是如下的一种形式：

“如果R的两个元组在属性 A_1, A_2, \dots, A_n 上一致（即，这些元组在各自的组件中对于这些属性中的每一个都具有相同的值），那么他们一定也在另一个属性B上一致。”

写为：

$A_1, A_2, \dots, A_n \rightarrow B$

R

	A_1	...	A_n					B	
t									
t'									

如果 t, t' 在这里一致 那么 t, t' 在这里也一致

异常

- 冗余
 - 不必要的重复教师信息
- 插入异常
 - 如果此人不教授课程怎么办？
无法插入新的教员
- 更新异常
 - 可能导致多个信息副本冲突
- 删除异常
 - 如果课程不再开设怎么办？
我们可能会丢失我们想保存的其他信息...

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

424 Dr. Newsome 29-Mar-2007 ?

Employees' Skills

Employee ID	Employee Address	Skill
426	87 Sycamore Grove	Typing
426	87 Sycamore Grove	Shorthand
519	94 Chestnut Street	Public Speaking
519	96 Walnut Avenue	Carpentry

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

DELETE

当我们试图把太多的东西塞进一个单一的关系表中时，出现的问题被称为异常。我们遇到的异常的主要类型是：

- 冗余：信息可能在几个元组中不必要地重复。
- 插入异常：要插入的实体的属性要求太多。
- 更新异常：更新时需要在多个地方更改信息。
- 删除异常：可能会丢失不想删除的数据。

Employees(EmpID, Name, Phone, Position)

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E1247	John	9876	Salesrep
E1597	Smith	9876	Salesrep
E5239	Mary	1234	Lawyer
E6411	Peter	1234	Lawyer

上述关系表中是否存在FD?

- EmpID决定一切
- Position决定Phone (extension)
- 但Phone不决定Position

示例：有多个电话号码的人

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield
Joe	987-65-4321	908-555-1234	Westfield

- PhoneNumber → everything
- SSN → Name, City

有什么异常吗？

- 冗余 = 重复数据
- 更新异常 = Fred 移动到 “Bellevue”
- 删除异常 = Fred 删除所有电话号码

慕课：什么是函数依赖

慕课视频片段

视频名称：Video



温馨提示：此视频框在点击“上传手机课件”时会进行转换，用手机进行观看时则会变为可点击的视频。此视频框可被拖动移位和修改大小

分解关系表

消除这些异常的方法是分解关系表。

将以前的关系分解为两个：

Name	<u>SSN</u>	City
Fred	123-45-6789	Seattle
Joe	987-65-4321	Westfield

<u>SSN</u>	<u>PhoneNumber</u>
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121
987-65-4321	908-555-1234

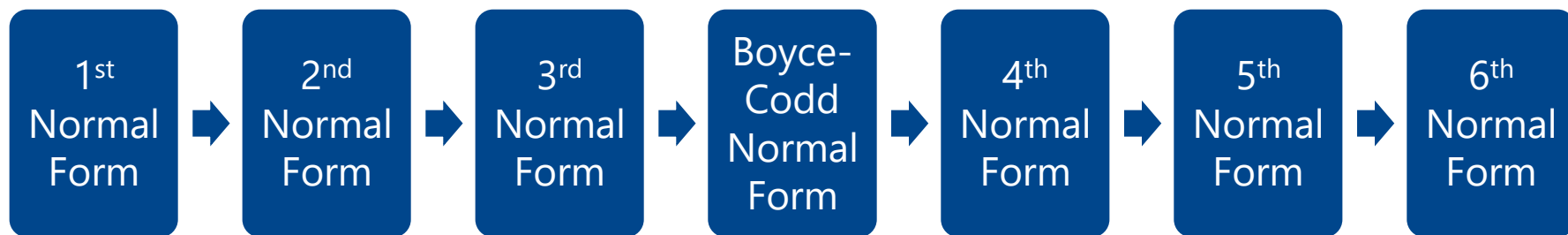
分解的目标是使用两个或多个不存在异常的关系来替换一个冗余的关系表。

规范化过程：

- 从一些关系模式开始
- 找出函数依赖 (FD)
- 使用它们来设计更好的关系表模式

关系模型的发明者Edgar Codd提出了范式理论

- 第一范式
- 扩展到第二范式和第三范式
- Boyce-Codd范式 (BCNF) : Raymond F.Boyce与Edgar-Codd的结合



Boyce-Codd范式

分解的目标是使用两个或多个不存在异常的关系来替换一个冗余的关系表。

有一个简单的条件可以保证不存在异常。 这种条件被称为：

Boyce Codd normal form 或 BCNF

Boyce-Codd范式

候选码(Candidate key):

- 可以唯一地标识单个记录的一组属性，但不一定指定为键

主属性 (Prime Attribute):

- 作为候选键的成员的属性

超码 (Super Key):

- 包含（候选）键的一组属性称为超级键，是“键的超集”的缩写。

平凡函数依赖(Trivial Functional Dependencies):

- 如果B是A中的一部分，则 $A_1, A_2, \dots, A_n \rightarrow B$ 是平凡的（例如，学生ID和学生姓名->学生姓名）

Boyce-Codd范式

Boyce-Codd范式 (BCNF) :

- 关系表 R 是符合 BCNF 范式, 当且仅当: 每当 R 有一个非平凡的FD $A_1, A_2, \dots, A_n \rightarrow B$, 那么 $\{A_1, A_2, \dots, A_n\}$ 是 R 的一个超码

也就是说, 每个非平凡FD的左边必须是一个超键。

Boyce-Codd范式

Class	Professor	EmployeeID
CEE 327	Anne Goodchild	12994
CEE 367	Pedro Arduino	12223
CEE 377	Gregory Miller	12889
CEE 410	Ryan Avery	12786
CEE 454	Ryan Avery	12786

什么是函数依赖关系？

- Class → Everything – Only 候选码
- EmployeeID → Professor – 非平凡函数依赖

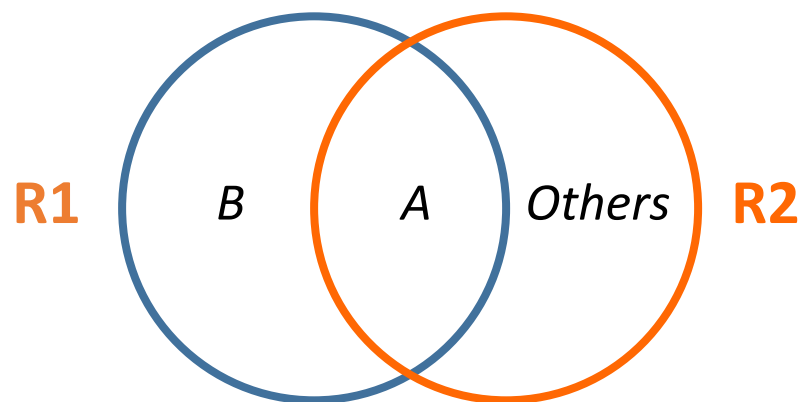
EmployeeID 不是一个超码

结论：此关系表不符合 BCNF

转换为BCDF

算法（针对每个关系表）：

1. 选取任何违反所选范式的函数依赖项 $A \rightarrow B$ 。
2. 拆分为 $R1(A, B)$ 和 $R2(A, \text{Rest})$ 。
3. 标识 $R1$ 和 $R2$ 的键。
4. 重复。



示例：分解以下描述零售商系统中采购的关系表。

Purchase (InvoiceNumber, Date, Time, CustomerID, CustomerName, StoreName, StoreAddress)

标识函数依赖：

1. InvoiceNumber \rightarrow Everything
2. Date, Time, CustomerID \rightarrow Everything
3. StoreAddress \rightarrow StoreName
4. CustomerID \rightarrow CustomerName

选择一个违反BCNF的 FD

4. $\text{CustomerID} \rightarrow \text{CustomerName}$

将 R 分解为两个关系表，并标识每个关系表的键。

$\text{Purchase}(\underline{\text{InvoiceNumber}}, \text{Date}, \text{Time}, \text{CustomerID}, \text{StoreName}, \text{StoreAddress})$
 $\text{Customer}(\underline{\text{CustomerID}}, \text{CustomerName})$

转换为BCDF

Purchase(InvoiceNumber, Date, Time, CustomerID, StoreName, StoreAddress)

Customer(CustomerID, CustomerName)

识别函数依赖

Purchase:

1. InvoiceNumber \rightarrow Everything
2. Date, Time, CustomerID \rightarrow Everything
3. StoreAddress \rightarrow StoreName

Customer:

4. CustomerID \rightarrow CustomerName

选择一个违反BCNF的FD

3.StoreAddress \rightarrow StoreName

分解关系表，并为每个创建的关系表标识键。

Purchase(InvoiceNumber, Date, Time, CustomerID, StoreAddress)

Customer(CustomerID, CustomerName)

Store(StoreAddress, StoreName)

所有关系都符合BCNF

其他范式

第一范式 (1NF) :

- 每个元组的每个组成部分都是原子属性

第二范式 (2NF) :

- 允许关系中的传递函数依赖 (FD) , 但禁止左侧为键的真子集的非平凡函数依赖。

第三范式 (3NF) :

- 关系表 R 符合第三范式 (3NF) : 当 $A_1, A_2, \dots, A_n \rightarrow B$ 是非平凡 FD 时, $\{A_1, A_2, \dots, A_n\}$ 是一个超码, 或者 B 是某个键的成员。

换句话说：

第二范式 (2NF)：

在满足1NF基础上，每个非主属性都依赖于每个候选键的全部。

第三范式 (3NF)：

在满足2NF基础上，每个非主属性都（非传递地）依赖于所有候选键的整体。

1NF: 表中的每个字段都是最小的数据单元

例如，用户信息中，“联系方式”字段不应该为“手机+居住地址”，而是单独作为两个原子值字段。否则当需要针对“手机”进行查询时，不管是查询性能和成本都是不理想的。

2NF: 在 1NF 的基础上，表中所有的非主属性必须完全依赖于候选码

因为我们知道在一个订单中可以订购多种产品，所以单单一个 OrderID 是不足以成为主键的，主键应该是 (OrderID, ProductID)。显而易见 Discount (折扣)，Quantity (数量) 完全依赖 (取决) 于主键 (OrderID, ProductID)，而 UnitPrice, ProductName 只依赖于 ProductID。所以 OrderDetail 表不符合 2NF。不符合 2NF 的设计容易产生冗余数据。

3NF: 在 2NF 的基础上，非主属性之间没有相互依赖 (消除传递依赖)

考虑一个订单表【Order】(OrderID, OrderDate, CustomerID, CustomerName, CustomerAddr, CustomerCity) 主键是 (OrderID)。其中 OrderDate, CustomerID, CustomerName, CustomerAddr, CustomerCity 等非主键列都完全依赖于主键 (OrderID)，所以符合 2NF。不过问题是 CustomerName, CustomerAddr, CustomerCity 直接依赖的是 CustomerID (非主键列)，而不是直接依赖于主键，它是通过传递才依赖于主键，所以不符合 3NF。

关系表示例

Class	Professor	EmployeeID
CEE 327	Anne Goodchild	12994
CEE 367	Pedro Arduino	12223
CEE 377	Gregory Miller	12889
CEE 410	Ryan Avery	12786
CEE 454	Ryan Avery	12786

- Class → Everything – Only candidate key
- EmployeeID → Professor – Non-trivial functional dependency
- Class → EmployeeID → Professor – Transitive FD (符合2NF)

Class本身能够成为一个主键,
一切都可以由Class决定



关系在2NF中

EmployeeID不是一个超码,
Professor不是一个候选码



关系不在3NF中

Boyce-Codd与第三范式

Boyce-Codd范式:

- 对于所有非平凡函数依赖项 $A \rightarrow B$, A 必须是一个超码

第三个范式:

- 对于所有非平凡函数依赖项 $A \rightarrow B$, A 必须是一个超码或者 B 是一个主属性

Prime attribute : 候选码的成员

- 3NF 不允许非主属性被另一个非主属性决定, 但允许主属性被主属性决定, 而 BCNF 中, 任何属性 (包括非主属性和主属性) 都不能被主属性所决定 (主属性内部也不能部分或传递依赖)

关系表示例

2NF: 每个非主属性都依赖于每个候选键的整体。

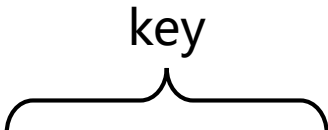
key		key		
CabinetID	Route	Milepost	RouteType	Type
1672211	SR167	22.11	State Route	Meter
0901501	I90	150.1	Interstate	Speed
0901512	I90	151.2	Interstate	Speed

Route Type 由 Route 决定

不符合2NF: 非主属性由键的子集决定的。

关系表示例

3NF: 对于所有非平凡函数依赖项 $A \rightarrow B$, A 必须是一个超码, 或者 B 是一个主属性。



AccidentID	Route	Milepost	Date	Season
1672211	SR167	22.11	1/12/2013	Winter
0901501	I90	150.1	5/29/2013	Summer
0901512	I90	151.2	11/19/2013	Winter



Season 由 Date 决定

符合 2NF: 一切 (直接或通过传递) 由 AccidentID 决定。

不符合 3NF: Date 不是一个超码并且季节不是一个主属性。

关系表示例

BCNF：对于所有非平凡函数依赖 $A \rightarrow B$ ， A 必须是一个超码。

Player	Number	Team	Stadium
Marshawn Lynch	24	Seahawks	CenturyLink
Peyton Manning	18	Broncos	Sports Authority
Russell Wilson	3	Seahawks	CenturyLink

有很多重叠的候选码：{Player}, {Number, Team}, {Stadium, Number}等，所以没有非主属性

但是：Team \rightarrow Stadium

符合3NF：Stadium 是一个主要属性。

不符合BCNF：Team不是超码。