数据库一天从入门到及格（下）

“末日时在做什么？有没有空？可以来学会儿数据库吗？”

Author:Yingluosanqian

第四章 高级数据库模型

4.1 E/R模型

在 *实体-联系（ER）* 模型中，数据的结构用图形化方式表示，用到以下三个主要的元素类型：

1. 实体集
2. 属性
3. 联系

4.1.1 实体集

实体（entity）是某种抽象对象，相似实体的集合形成实体集（entity set）。**在E/R图中用矩形表示**

4.1.2 属性

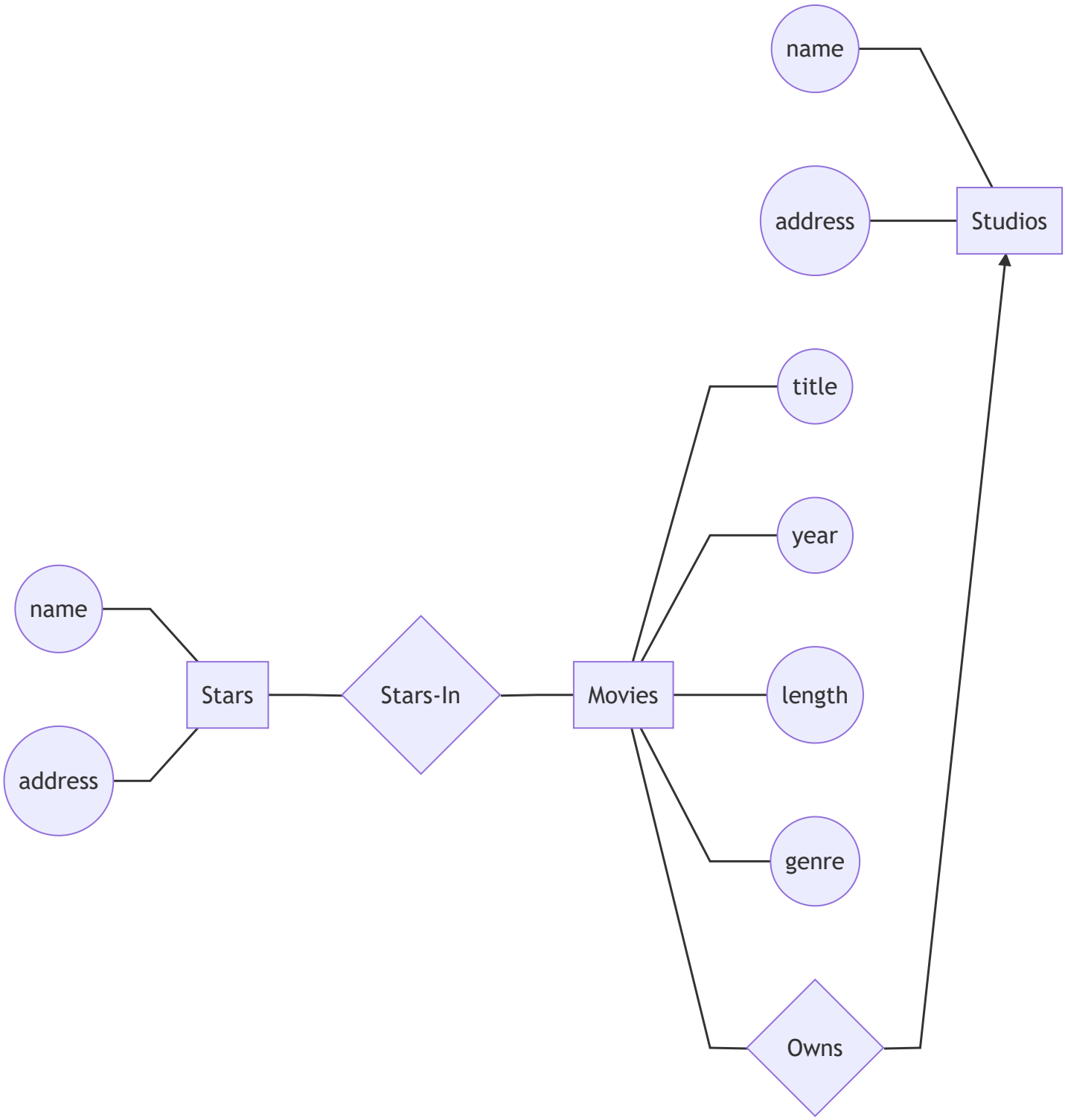
实体集有相关的属性（attribute）属性是这个实体集中的实体具有的性质。**在E/R图中用圆形表示**

4.1.3 联系

联系（relationship）是两个或多个实体集的连接。**在E/R图中用菱形表示**

4.1.4 实体-联系图

E/R图是描述实体集、属性和联系的图示。途中每种元素都用节点表示并且使用特殊形状的节点来标识特定的类型。



图中指向Studios的箭头暗示每部电源只属于唯一的电影公司。

4.1.5 E/R图实例

E/R图是一种描述数据库模式的符号。一个用E/R图描述的数据库包含特定的数据，成为数据库实例。

4.1.6 二元E/R联系的多样性

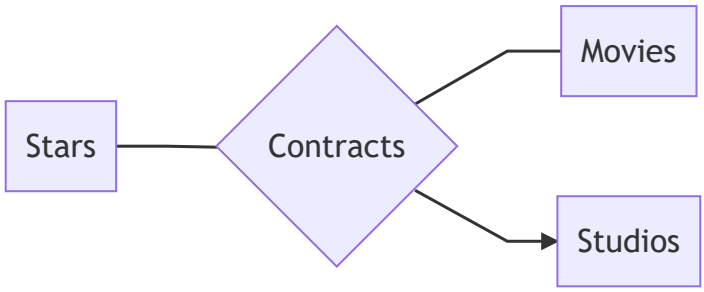
假设R连接实体E和F，那么：

1. 如果E中的任一实体可以通过R与F中的至多一个实体联系，那么说R是从E到F的多对一联系。
2. 如果R既是从E到 F 的多对一联系，又是从F到E的多对一联系，那么R就是一对一联系。
3. 如果R既不是从E到 F 的多对一联系，也不是从F到E的多对一联系，那么R就是多对多联系。

注： 如果实体集E到F是多对一联系，就把箭头指向F。

4.1.7 多路联系

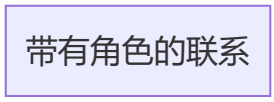
例如：

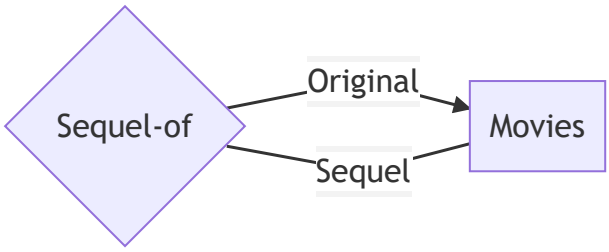


注： 在多路联系中，指向实体集E的箭头表示：如果从该联系的其他每个实体集中选择一个实体，它们至多与E中的一个实体联系。

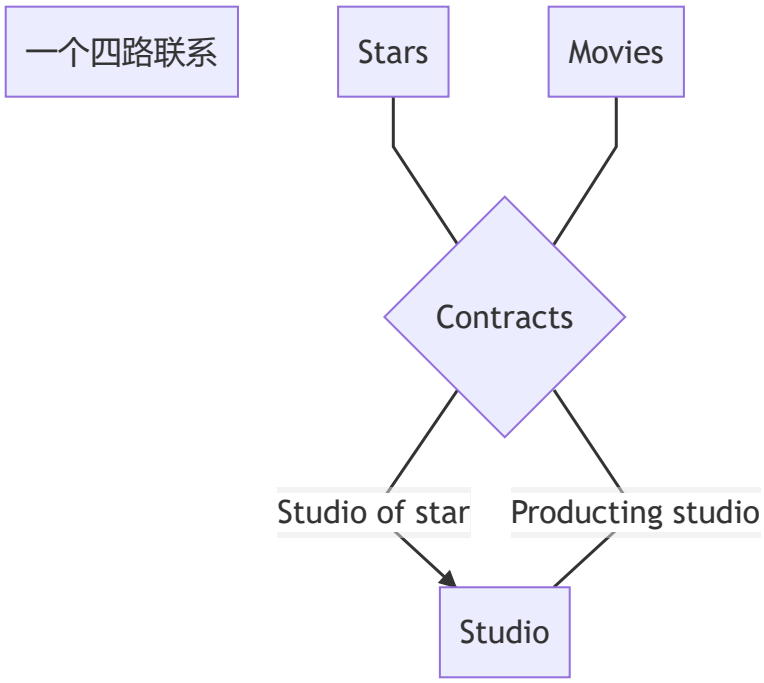
4.1.8 联系中的角色

在一个联系中一个实体集可能出现多次。如果是这样，根据实体集在联系中出现的次数，把联系与实体集用同样多的连线连起来。每一条连向实体集的连线代表实体集在练习中扮演的不同角色。**因而，人们给实体集和联系之间的边命名，称之为“角色”。**





(关于电影前作与续集



(关于影星的属于与借用

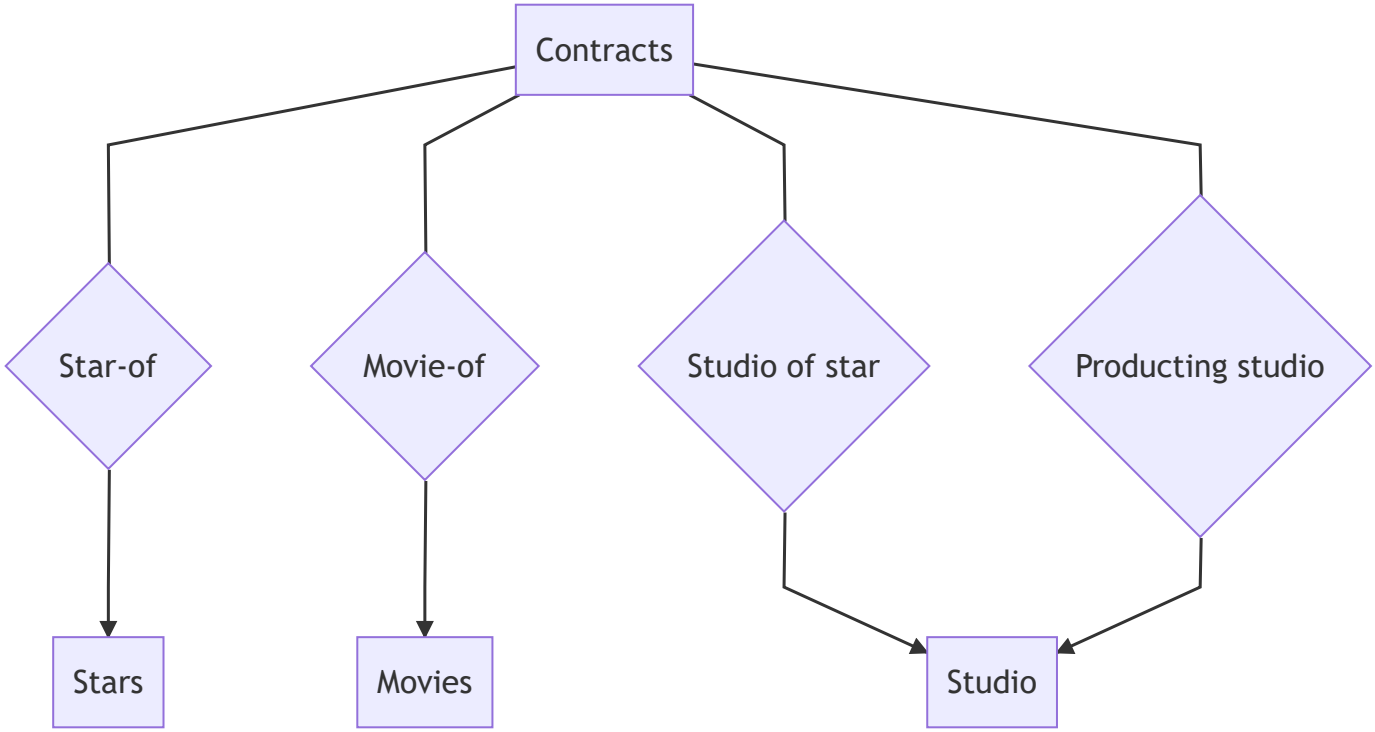
4.1.9 联系的属性

通常联系上可以防止一个或者更多的属性。（书P78）

4.1.10 多路联系到二元联系的转换

虽然E/R模型不限制联系是二元联系，但是有必要看一下连接多个实体集的联系是怎样转化为一组二元的多对一联系。

为此，需要引入一种实体集，它的实体被看作是多路联系的联系集的元组。这个实体集叫做连接实体集。



4.1.11 E/R模型中的子类

简单来说，一个实体集的子类自带父类的属性（并继承父类的联系），并且子类还可以有它独有的属性。我们用一个被称作isa的联系连接实体集和它的子类。

注： 尽管没有标出其一对一联系的两个箭头，但每个isa联系都是一对一联系。

注： 不熟悉画图，就不画了。

4.2 涉及原则

本节内容建议自行查看例子理解，（因为我也不会

4.2.1 忠实性

4.2.2 避免冗余

4.2.3 简单性

4.2.4 选择正确的联系

4.2.5 选择正确的元素种类

4.3 E/R模型中的约束

E/R模型有几种方式来表达构建数据库过程中数据上的约束。像关系模型一样，可以通过属性表示实体集的键。

4.4.1 E/R模型中的键

实体集的键是有一个或多个属性的集合K，对于来自E的不同实体e_1和e_2，它们对键K中的属性没有完全相同的值。

注意以下几点：

1. 每个实体集都必须有一个键。（尽管在isa层次和“弱”实体集情况下，键实际上属于另一个实体集）
2. 一个实体集也可以有多个键。（但习惯上只选择一个键作为“主键”，就好像只有一个键一样）
3. 当一个实体集出于一个isa层次中时，要求根实体集拥有键所需的所有属性，并且每个实体集的键都可以在跟实体集中发现它们的组成部分，而不管在这个层次中有多少个实体集是它的组成部分。

4.3.2 E/R模型中键的表示

在E/R图中，一个实体集的键用下划线标出。

注意： 在一些反常情况下，构成一个实体集的键的属性并不完全属于它自己。这叫做“弱实体集”。

4.3.3 引用完整性

“引用完整性”约束是说一段在上下文中出现的值一定在另一端上下文中出现。

扩展E/R图的箭头标记可以用来表示一个联系是否被期望在一个或多个方向上支持引用完整性。假设R是从实体集E到实体集F的联系。**用圆箭头指向F表示此联系从E到F不仅是多对一或一对一，而且要求与给定的E实体相联系的F实体必须存在。**

4.3.4 度约束

在E/R模型中，可以在连接一个联系到一个实体集的边上加一个数字，表示相关实体集中任意可被联系到的实体数目的约束。（不难理解吧）

4.4 弱实体集

一个实体集的键是由另一个实体集的部分或全部属性构成，这样的实体集叫做弱实体集（weak entity set）。

4.4.1 弱实体集的来源

1. 有时实体集处在一个isa层次分类无关的层次体系中。如果集合E中的实体是集合F中实体的一部分，那么可能仅仅只考虑E实体的名字将不具有唯一性，需要再考虑了E实体所属的F实体的名字后

唯一性才成立。
2. 弱实体集来自连接实体集，它被作为消除多路联系的一种方法。

（听着很晕，我也没搞懂）

4.4.2 弱实体集的要求

弱实体集的键属性不加选择将不能得到。相反，如果E是弱实体集，则它的键组成为：

1. 零个或多个它自己的属性；
2. 从E到其他实体集的多对一联系连接的键属性。（并且这些多对一联系成为E的**支持联系** 。为了使E到某个实体集F的多对一联系成为E的一个支持联系，必须付出下面的条件：
    1. R必须是从E到F的二元的多对一联系；
    2. R必须有从E到F的引用完整性；
    3. F提供给E作键的属性必须是F的键属性；
    4. 如果F本身就是弱实体集，那么F提供给E的F的部分或全部的键属性是由支持联系连接的一个或多个实体集G的键属性，如此递归。
    5. 如果E到F有多个不同的支持联系，那么每个联系被用来提供一份F的键的拷贝以帮助E键的形成。（这条讲的是个啥样啊QuQ）

4.4.3 弱实体集的符号

用下面的约定来描述一个实体集是弱实体集，并且声明它的键属性：

1. 如果一个实体集是弱实体集，它就被显示为**双边的矩形**。
2. 它支持的多对一联系显示为**双边的菱形**。（并非所有从弱实体集连接出去的联系都是支持联系）
3. 如果一个实体集为它自己的键提供了任何属性，那么那些属性就带有下划线。

4.5 从E/R图到关系设计

emmmm书上讲的我看不懂，于是我又重构了本节内容.....（也许会和书上的有出入？

4.5.1 基本操作

1. 实体集的转换
    - 一个实体集转换为一个关系模式；
    - 实体的一个属性对应该表的一个列；
    - 实体的主键就是表的主键。
2. 联系的转换
    - 一个联系转换为一个关系模式；
    - 联系的属性对应表的属性，另外并上所有参与联系的各实体主键的并集。
    - 主键的设置

    - 如果联系时多对多，主键是所有参与联系的各实体的主键的并集；
    - 如果联系是多对一，主键是多端实体的主键；
    - 如果联系是一对一，主键是任一端实体的主键。
3. 主键相同的关系模式可以进行合并。（嗯，就是字面意思，常见于某个联系转换的关系模式与某个实体集转换的关系模式的合并）

4.5.2 处理弱实体集

P.S. 下面给出两种描述，一种是书上的，一种是其他地方找的，我认为两种描述是一致的。

书上的描述

若W是一个弱实体集，则W转化为关系后的模式组成为：

1. W的**所有属性**
2. 与W相连的**支持联系的所有属性**
3. 对每一个连接W的至此联系，即从W到实体集E的多对一联系，要包含**E的所有关键字属性**。（若重名可重命名）
4. 不要为W的支持联系构造关系。

另一个描述

1. 一个关联弱实体的联系和弱实体一起转换为一个关系表（等价上述第四点）
2. 弱实体**属性集**、关联弱实体的支持联系的**属性集**、依赖的强实体**主键**，并集就是表的**属性集**（等价上述第1，2点）
3. 如果弱实体集在联系的基数上处于**多端**，主键是**参与联系的强实体主键并上弱实体的标识主键**。（等价上述第三点）
4. 如果弱实体集在联系的基数上处于**一端**，主键是**参与联系的强实体主键**。（这一点建议和本描述第三点对比，为什么不需要并上弱实体的标识主键了？因为不需要了！）

思考

- 为什么不需要为支持联系构造关系？因为根据基本操作中提到的 **“主键相同的关系模式可以进行合并”**，支持联系转换的关系就已经合并到弱实体集的联系里了。
- 规则这么多要怎么记忆？在规则中把握三个要点：**转换为关系的是什么？哪些东西作为属性？哪些东西作为主键？** 把握以上三点，再思考为什么是这些东西，问题就迎刃而解了。

4.6 子类结构到关系的转化

有三种策略，下面将逐一介绍。（有点抽象……总之我看着很迷）

4.6.1 E/R方式转化

同往常一样给每个实体集建立一个关系。如果实体集E不是层次中的根，则为了能够区别元组表示的实体，关系E要包含根的键属性和E的本身属性。并且，如果E和其他实体集有联系，就使用这些键属性标识与此联系对应的关系中的E实体。

我的理解： 也就是说就用直接约定的方法就好了。

4.6.2 面向对象方法

把isa-层次转化为关系的方法就是枚举层次中 **（根节点）的所有可能的子树**，为每一个子树构造一个可以描述该子树中实体的关系，这个关系模式含有子树中所有实体集的所有属性。（前提是isa-层次满足树形结构，更准确地说：除了根的每个实体有且仅有一个父类）

我的理解： 所有可能的子树就对应了所有的情况，所以这个方法相当于进行了一个大分类。以书P98页图举例，可分为四类：并不指明具体类型的电影类，卡通类的电影，凶杀类的电影，既是卡通又是凶杀类的电影。

4.6.3 使用空值组合关系

如果允许元组中有NULL的话，就可以对一个实体集层次只创建一个关系。这个关系把含了层次中所有实体集的所有属性。一个实体就表现为关系中的一个元组。元组中的NULL表示该实体没定义的属性。（这也太暴力了趴）

我的理解： 没啥好理解的，很暴力也很显然的一个方法。

4.6.4 各种方法的比较

我的建议： 下面给出了某些情况下各种方法的优劣，不妨思考：我们真的明白了为什么这个方法更优吗？如果相通了这个问题，那对于上述三种方法的理解及其蕴含的思想也将加深。

使用的例子为书P98页的图。

1. 总所周知涉及及格关系的查询代价昂贵（指耗时），所以人们宁愿在一个关系中寻找查询需要的所有属性，**“空值”方法在这一点上有着很好的性能。**
2. “2008年的哪几部影片放映时长长于150min呢？”，不难发现这个询问使用E/R方法转换来的关系可以很快得到，然鹅若使用面向对象方法转换的关系就痛苦面具了（这是因为它要在一堆表（当然是各种可能子树的表辣）里面查）。
3. “放映长于150min的卡通片使用了什么武器？”，这一类查询就更适合面向对象方法而不是E/R方法。因为面向对象方法中可以直接去有武器的子树中找>150min的。
4. 如果倾向于使用尽量少的关系，就可以使用“空值”方法，因为它只需要一个关系。
5. 有时需要减少空间和避免重复的信息，这时候“空值”方法就不行了，而E/R方法由于关键字的重复次之，面向对象方法最优（因为每一个实体仅使用一个元组）。

第五章 代数和逻辑查询语言

声明：这里仅仅叙述老师上讲的部分。

5.1 包上的关系操作

什么是包？**包（bag）是一个多重集（multiset）而不是集合（set）。** 更通俗地说，包中一模一样的元组可以多次出现，但是这在关系中是不可以的，关系中的元组是去重的。

5.1.1 为什么采用包

根本原因：**效率更高 AND 能处理一下特殊情况**

1. 效率更高是因为不需要去重了。（如果你熟悉程序设计，就会知道去重的复杂度大概是O(nlogn)的（吧））
2. 比如要求一群人的平均年龄，如果有年龄相同的年龄被去重了，那求出来的就不是平均年龄了，**这就是一种特殊情况。**

5.1.2 包的交、并、差

首先，**定义包内元素的次数为这个元素在包内出现的次数。**

然后，逐一讨论交、并、差，假设元组t在包R中出现n次，在包S中出现m次。

1. 在$R \cup S$操作中，元组t出现n+m次。
2. 在$R \cap S$操作中，元组t出现min(n,m)次。
3. 在$R - S$操作中，元组t出现max(0,n-m)次。

5.1.X 注意

包的操作类似于关系，唯一的不同在于包并不去重，掌握了这一点，接下来的内容都很显然了。

5.1.3 包的投影操作

5.1.4 包的选择操作

5.1.5 包的笛卡尔积

5.1.6 包的连接

5.4 关系代数与Datalog

我看不懂，老师是不是跳过了什么内容.....（也许没讲吧）

第六章 数据库语言SQL（sequel）

这些网站可以拿来练手

这里有现成的数据表
这里支持DIY从头造表

建议看书P143-P183，生硬的语法知识和一些嵌套小技巧。

（爬了爬了...

第七章 约束与触发器

7.1 键与外键

7.1.1 外键约束声明

格式：REFERENCES <表名> (<属性名>)

```
CREATE TABLE Studio(
    name    CHAR(30)    PRIMARY KEY,
    address VARCHAR(255),
    presC  INT  REFERENCES  MovieExec(cert)
);
```

```
CREATE TABLE Studio(
    name    CHAR(30)    PRIMARY KEY,
    address VARCHAR(255),
    presC  INT,
    FOREIGN KEY (presC)  REFERENCES  MovieExec(cert)
);
```

7.1.2 维护引用完整性

- DBMS会阻止如下错误行为
  - 对Studio插入一个元组，**其presC值非空**，且它不是MovieExec关系中任何元组的certC值。
  - 修改Studio中一个元组，**其presC值非空**，且它不是MovieExec关系中任何元组的certC值。
  - 删除MovieExec元组，而该元组的certC值非空且是一个或多个Studio元组的presC值。
  - 修改MovieExec元组的certC值，而旧的certC值非空且是一个或多个Studio元组的presC值。
- 对于上述后两种错误行为，设计者可以在以下三种选项中进行选择
  - 缺省原则（THE DEFAULT POLICY）：拒绝违法更新。

- 级联原则（THE CASCADE POLICY）：被引用属性组的改变被仿造到外键上。（被引用属性组修改，则外键修改为同样的值，被引用属性组被删除，则外键所在元组同样删除）
- 置空值原则（THE SET NULL POLICY）：被引用关系上的更新影响其外键值时，后者被改为空值（NULL）。

```
CREATE TABLE Studio(
    name CHAR(30) PRIMARY KEY,
    address VARCHAR(255),
    presC INT REFERENCES MovieExec(cert)
        ON DELETE SET NULL
        ON UPDATE CASCADE
);
```

- 循环约束，比如表A里的a属性引用表B里的b属性，而表B里的b属性也引用表a里的a属性。这样，永远无法先设置a属性或b属性的值，因为这会违反规则。因此，引入DEFERRABLE保留字，使得外键约束的检查推迟，在这之前先完成设置a属性和设置b属性。DEFERRABLE INITIALLY DEFERRED表示延迟检查，DEFERRABLE INITIALLY IMMEDIATE表示立即检查。

```
CREATE TABLE Studio(
    name CHAR(30) PRIMARY KEY,
    address VARCHAR(255),
    presC INT UNIQUE
        REFERENCES MovieExec(cert)
        DEFERRABLE INITIALLY DEFERRED
);
```

7.2 属性和元组上的约束

SQL的CREATE TABLE语句中可以声明两种约束。

- 在单一属性上的约束
- 在整个元组上的约束

7.2.1 非空值约束

NOT NULL：不允许属性取空值

```
//Studio中的presC不允许取空值
presC INT REFERENCES MovieExec(cert) NOT NULL
```

- 如此声明后，SET NULL原则将不能使用

7.2.2 基于属性的CHECK约束

```
presC INT REFERENCES MOvieExec(cert)
    Check(presC >= 100000)

gender CHAR(1) CHECK(gender IN('F','M'))


presC INT CHECK
    (presC IN(SELECET cert FROM MovieExec))
```

7.2.3 基于元组的CHECK约束

```
CREATE TABLE Studio(
    //
    ......
    //
    CHECK(gender = 'F' OR name NOT LIKE 'MS.%')
);
```

7.2.4 基于元组和基于属性的约束的比较

- 如果元组上的约束涉及该元组的多个属性，则必须使用基于元组的CHECK约束。
- 如果仅涉及一个属性，那么可以作为基于元组或基于属性的约束。

7.3 修改约束

任何时候都可以添加、修改、删除约束。

7.3.1 给约束命名

为了修改或删除一个已经存在的约束，约束必须有名字。为了命名在约束前加保留字CONSTRAINT和该约束的名字。

```
name CHAR(30) CONSTRAINT NameIsKey PRIMARY KEY

gender CHAR(1) CONSTRAINT NoAndro
    Check (gender IN('F','M'))

CONSTRAINT RightTitle
    Check (gender = 'F' OR NOT LIKE 'MS.%')
```

7.3.2 修改表上的约束

- 通过SET CONSTRAINT语句，可以将约束检查设置为延期检查。

```
SET CONSTRAINT MyConstraint DEFERRED;
```

- 删除约束

```
//删除约束
ALTER TABLE MovieStar DROP CONSTRAINT NameIsKey;
ALTER TABLE MovieStar DROP CONSTRAINT NoAndro;
ALTER TABLE MovieStar DROP CONSTRAINT RightTitle;
```

- 新增约束

```
//新增约束
//这些约束均基于元组而非基于属性
ALTER TABLE MovieStar ADD CONSTRAINT NameIsKey
    PRIMARY KEY(name);
ALTER TABLE MovieStar ADD CONSTRAINT NoAndro
    Check (gender IN('F','M'));
ALTER TABLE MovieStar ADD CONSTRAINT RightTitle
    Check (gender = 'F' OR NOT LIKE 'MS.%')
```

7.4 断言

- 断言和触发器时数据库模式的一部分，等同于表。
- 断言时SQL逻辑表达式且总是为真。
- 触发器时与某个事件相关的一系列动作，例如向关系中插入元组。触发器总是当这些事件发生时被执行。

7.4.1 创建断言

形式：CREATE ASSERTION <断言名> CHECK (<条件>)

- 任何引起断言条件为假的数据库更新都被拒绝。
- 在某些情况下，在涉及子查询时，CHECK约束违反时，可以在某些条件下避免操作被拒绝。（如7.2.2第二条，若删除电影公司经理元组，此变化对上述CHECK约束不可见。）

7.4.2 使用断言

- 断言条件中引用的任何属性都必须要介绍，特别是要体积在select-from-where表达式中的关系。

```
CREATE ASSERTION RichPres CHECK(
    NOT EXISTS(
        SELECT Studio.name
        FROM Studio,MovieExec
        WHERE presC = cert AND netWorth < 10000000>
    )
);

CREATE ASSERTION RichPres CHECK(
    10000 >= ALL(
        SELECT SUM(length)
        FROM Movies
        GROUP BY studioName
    )
);
```

- 同样地，断言可以被删除。

7.5 触发器

- 仅当数据库程序员声明的事件发生时，触发器被激活。
- 当触发器被事件激活时，触发器测试出发的条件。如果条件不成立，则响应该事件的触发器不做任何事情。
- 如果触发器声明的条件满足，则与该触发器相连的动作由DBMS执行。

7.5.1 SQL中的触发器

- 触发器的条件检查和触发器的动作可以在触发事件执行之前的数据库状态上或在触发动作被执行后的状态上执行。
- 条件和动作可以引用元组的旧值和/或触发事件中更新的元组的新值。
- 更新事件可以被局限到某个特定的属性或某一些属性。
- 程序员可以选择动作执行的方式。
    - 一次只对一个更新元组；
    - 一次针对在数据库操作中被改变的所有元组。

```
CREATE TRIGGER NetWorthTrigger
//创建语句
AFTER UPDATE OF networth ON MovieExec
//指明在触发事件之后动作
REFERENCING
    OLD ROW AS OldTuple,
    NEW ROW AS NewTuple
//命名
FOR EACH ROW
    WHEN  (OldTuple.netWorth>NewTuple.netWorth)
    BEGIN
        UPDATE MovieExec
        SET netWorth = OldTuple.newWorth
        WHERE cert = NewTple.cert;
        //......;//
    END;
```

- AFTER可用BEFORE替换，意义显然。
- UPDATE可用INSERT或DELETE替换，意义显然
- WHEN可选
- BEGIN,END之间的语句用分号分隔
- 修改操作中用OLD ROW AS和NEW ROW AS分别命名
- 默认为FOR EACH STATEMENT，此时FOR EACH ROW行级触发器，而是语句级触发器。
- 在语句级触发器中，不能用OLD ROW AS OldTuple。而不论是语句级还是行级触发器，都可以用OLD TABLE AS OldStuff和NEW TABLE AS NewStuff。

第八章 视图与索引

8.1 虚拟视图

视图，类似于表，是虚拟的。

8.1.1 视图定义

CREATE VIEW <视图名> AS <视图定义>;

```
CREATE VIEW ParamountMovies AS
    SELECT title,year
    FROM Movies
    WHERE studioName = 'Paramount';
```

```
CREATE VIEW MovieProd AS
    SELECT title,name
    FROM Movies,MovieExec
    WHERE productC=cert;
```

8.1.2 视图查询

- 把它当作表那样用就可了，没有什么额外要注意的。

8.1.3 属性重命名

```
CREATE VIEW ParamountMovies(movieTitle,prodName) AS
    SELECT title,year
    FROM Movies
    WHERE studioName = 'Paramount';
```

8.2 视图更新

8.2.1 视图删除

```
DROP VIEW ParamountMovie;
```

- 删除视图不会影响原TABLE，但是删除原TABLE视图就不可用了

8.2.2 可更新视图