

数据库系统原理笔记

陈鸿峥

2019.12*

目录

1	数据库系统概述	2
1.1	高层概述	2
1.2	数据库语言	2
1.3	关系型数据库	3
2	SQL简介	4
2.1	数据定义语言(DDL)	4
2.2	数据查询语言	5
3	进阶SQL	8
3.1	内外连接	8
3.2	视图	9
3.3	事务	9
3.4	完整性约束	9
3.5	授权	10
4	形式化关系查询语言	10
4.1	关系代数	10
4.2	其他关系演算	11
5	E-R模型	11
5.1	实体-关系模型	11
5.2	扩展的E-R属性	14
6	关系型数据库设计	14

*Build 20191227

本课程采用书目Avi Silberschatz, Henry F. Korth, S. Sudarshan, *Database System Concepts (6th ed)*¹。

1 数据库系统概述

1.1 高层概述

早期的数据库直接建立在文件系统上，但这会导致：

- 数据冗余与不一致
- 访问数据非常麻烦
- 数据孤立：数据分散在不同文件中，文件又有不同格式
- 完整性问题：难以添加限制（如年龄为非负整数）
- 原子性问题：要么全部发生要不根本不发生
- 多用户的并发访问
- 安全性问题：权限

数据抽象包括：

- 物理层：存储块、物理组织
- 逻辑层：通过类型定义进行描述，同时记录类型之间的相互关系
- 视图层：屏蔽数据类型细节的一组应用程序，同时提供了访问权限

查询过程：解释编译+求值(evaluation)

数据模型：

- 关系模型
- 实体-联系模型(ER)
- 基于对象的数据模型
- 半结构化数据模型：XML

1.2 数据库语言

- 数据操纵语言(Data Manipulation Language, DML)
- 数据定义语言(Data Definition Language, DDL)：DDL的输出放在数据字典中，数据字典包含元数据(metadata)，需要满足一致性约束

— 域约束：如整型、字符型等

¹<http://www.db-book.com>

- 参照完整性(referential integrity): 一个关系中给定属性集上的取值也在另一关系的某一属性集的取值中出现, 如 course记录中的dept_name必须出现在department关系dept_name属性中
- 断言(assertion): 域约束和参照完整性只是断言的特殊形式, 如“每一学期每一个系必须至少开设5门课程”
- 权限(authorization)

1.3 关系型数据库

1.3.1 结构

每一个表(table)都是一个关系(relation), 纵向为属性(attributes/columns), 横向为元组(tuples/rows)。一组特定的行称为关系实例(instance)。

ID	name	dept_name
12345	Chen	CS
10001	Bob	Biology
10101	Alice	Physics

注意关系都是无序的, 元组可以以任意顺序存储。

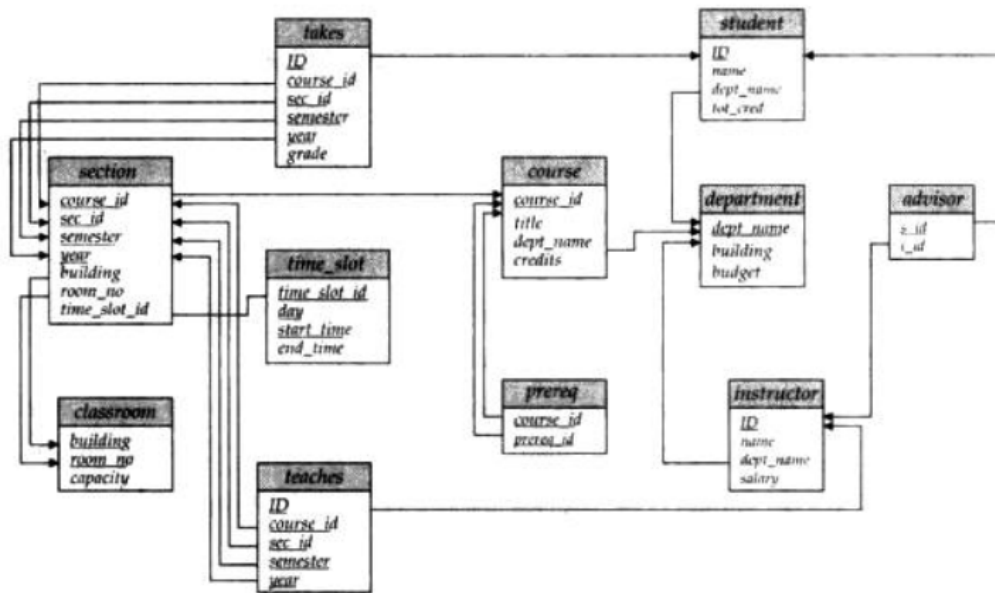
数据库模式(schema)是数据库的逻辑设计, 如instructor(ID,name,dept_name,salary) (可以理解为一个抽象); 数据库实例(instance)则是给定时刻数据库中数据的一个快照, 与具体数据相关。

1.3.2 码

一个元组的属性值必须是能**唯一**区分元组的, 即一个关系中没有两个元组在所有属性上取值都相同。

- 超码(superkey): 一个或多个属性的集合使得在一个关系中可以唯一地标识一个元组, 如ID属性是超码, 而名字不是
- 候选码(candidate key): 最小超码, 即其任意真子集都不能成为超码
- 主码(primary key): 用在一个关系中区分不同元组的候选码, 选择要慎重, 哪怕每个人只有一个地理地址, 但是也不应该用其当作主码
- 外码(foreign key): 一个关系模式 r_1 可能在它的属性中包括另一个关系模式 r_2 的主码, 则这个属性在 r_1 上称作参照 r_2 的外码。关系 r_1 也被称为外码依赖的参照关系, r_2 叫外码的被参照关系。

含有主码和外码以来的数据库模式可用模式图(schema diagram)表示。每个关系用一个矩形表示, 关系名字显示在矩形上方, 矩形内列出各属性, 主码属性用下划线标注, 外码依赖用箭头表示。



2 SQL简介

SQL即结构化查询语言(Structured Query Language, SQL)

2.1 数据定义语言(DDL)

数据类型

- char(n): 固定长度字符串
- varchar(n): 可变长度字符串, 用得最多
- int
- smallint
- numeric(p,d): 定点数, 共 p 位 (包括符号位), 其中 d 位在小数点右侧
- real, double precision
- float(n): 精度至少为 n 位的浮点数

```
create table instructor (
  ID char(5),
  name varchar(20) not null,
  dept_name varchar(20),
  salary numeric(8,2) default 0,
  primary key (ID), -- integrity constraints
  foreign key (dept_name) references department);
```

2.2 数据查询语言

基本的形式如下所示

```
select A1, A2, ..., An
from r1, r2, ..., rm
where P
```

其中 A_i 为属性(attribute)、 R_i 为一个关系(relation)/一个表、 P 是谓词(predicate)，返回满足关系的数组。

- 自然连接(natural join): 将两个关系的各个元组进行连接，并且只考虑那些在两个关系模式中都出现的属性上取值相同的元组对（相当于取交）
- 连接(join): 只考虑 A 这个属性进行连接

```
select name, title
from (instructor natural join teachers) join course using (course_id);
-- only join two relations with the same course_id
```

字符串运算：模式匹配

- upper(s)、lower(s)
- %: 匹配任意字符串
- _: 匹配任意一个字符
- like: 以这些字符串进行匹配

```
select distinct dept_name
from instructor

select * -- all attributes
from instructor

select '437' as F00

select ID, name, salary/12 as monthly_salary

select name
from instructor
where dept_name = 'Comp. Sci.' and salary > 80000

select *
from instructor, teaches -- Cartesian product

select distinct T.name
from instructor as T, instructor as S -- rename, Cartesian product
where T.salary > S.salary and S.dept_name = 'Comp. Sci.'
```

```

-- salary between 90000 and 100000

/**
percent ( % ). The % character matches any substring.
underscore ( _ ). The _ character matches any character.
**/

select name
from instructor
where name like '%dar%' matches any string containing "dar" as a substring

select distinct name
from instructor
order by name -- sorted

select *
from instructor
order by salary desc, name asc;

(select course_id from section where sem = 'Fall' and year = 2017)
union -- intersect, except
(select course_id from section where sem = 'Spring' and year = 2018)

select name
from instructor
where salary is null

-- aggregate functions
-- arg, min, max, sum, count
select dept_name, avg (salary) as avg_salary
from instructor
where dept_name= 'Comp. Sci.'
group by dept_name;

select count (distinct ID)
from teaches
where semester = 'Spring' and year = 2018;

-- group constraints
-- find the average salary in each dept
select dept_name, avg(salary) as avg_salary
from instructor
group by dept_name -- can only be contained by select clause
having avg(salary) > 42000; -- effected after grouping

```

三值逻辑，添加了Unknown

AND	OR	NOT
$T \wedge U = U$	$T \vee U = T$	
$F \wedge U = F$	$F \vee U = U$	$\neg U = U$
$U \wedge U = U$	$U \vee U = U$	

SQL在谓词中使用null测试空值，因而为找出instructor关系中salary为空值的所有教师，可写

```
select name
from instructor
where salary is null
```

集合成员资格通过in来判断，同时有some和all的关系测试，下面给出了一些嵌套子查询的例子。

```
select name
from instructor
where salary > some (select salary
                      from instructor
                      where dept_name = 'Biology')

select S.ID, S.name
from student as S
where not exists ((select course_id
                  from course
                  where dept_name = 'Biology')
                except
                (select T.course_id
                 from takes as T
                 where S.ID = T.ID))

select T.course_id
from course as T
where 1 >= (select count(R.course_id)
           from section as R
           where T.course_id = R.course_id and R.year = 2009);
-- there is no unique in MySQL

with max_budget(value) as
  (select max(budget)
   from department)
select budget
from department, max_budget
where department.budget = max_budget.value;

select dept_name,
```

```
(select count(*)
from instructor
where department.dept_name = instructor.dept_name)
as num_instructors
from department; -- scalar subquery
```

数据库的修改操作如下

```
delete from instructor
where dept_name = 'Finance';

insert into course(course_id, title)
values ('CS-437', 'Database Systems');

update instructor
set salary = salary * 1.05
where salary < 70000;

update instructor
set salary = case
    when salary <= 1000 then salary * 1.05
    when salary <= 2000 then salary * 2.05
    else salary * 1.03
end
```

3 进阶SQL

3.1 内外连接

on条件可以提供比自然连接更为丰富的连接条件

```
select *
from students join takes on student.ID = takes.ID; -- condition
```

直接使用natural join可能导致元组丢失，比如有一些学生没有选修任何课程，则其在student关系中不会与takes关系中任何元组配对。因此有外连接：

- 左外连接(left outer join)：只保留出现在左外连接运算之前（左边）关系中的元组，若右侧关系中没有对应属性则用null代替
- 右外连接(right outer join)
- 全外连接(full outer join)：左外连接和右外连接的组合

从而可以很容易查询出“所有课程一门也没有选修的学生”：

```
select ID
from student natural left outer join takes
```



```
where course_id is null;
```

为了将常规连接和外连接区分开，SQL中将常规连接称为内连接(inner join)，默认的都是内连接。

3.2 视图

让用户看到整个逻辑模型显然是不合适的，出于安全考虑，可能需要向用户隐藏特定的数据。本来通过select可以把需要的数据计算并存储下来，但一旦底层数据发生变化，查询的结果就不再匹配。因此，为了解决这样的问题，SQL提供一种虚关系，称为视图(view)，只有在使用的時候才会被计算。

```
create view faculty as
select ID, name, dept_name
from instructor;
```

之后便可以直接使用from语句访问视图。

定义 1 (物化视图(materialized view)). 如果用于定义视图的实际关系改变，视图也会跟着修改。

由于视图并不是数据库底层的关系，故一般数据库不允许对视图关系进行修改。

3.3 事务

SQL标准规定当一条SQL语句被执行，就隐式开始了一个事务。下列SQL语句之一会结束一个事务：

- Commit work: 将事务所做的更新在数据库中持久保存。在事务被提交后，一个新的事务自动开始。
- Rollback work: 撤销该事务中所有SQL语句对数据库的更新，数据库将恢复到执行该事务的第一条语句之前的状态。如果遇断电，回滚会在下一次重启时自动执行。

3.4 完整性约束

单个关系上的约束

- not null: 跟在属性定义后面
- unique (A_1, A_2, \dots, A_m): 声明 A_1, \dots, A_m 构成一个候选码
- check(<predicate>): 关系中每个元组都必须满足该谓词，如check(budget>0)

参照完整性/子集依赖：外码

```
foreign key (dept_name) references department
on delete cascade
on update cascade
```

会级联删除或更新。

3.5 授权

grant授权, revoke回收

- select: 允许用户修改关系中任意数组
- insert
- delete

用户名public包括了当前所有用户和未来用户的权限。

```
grant <authorization list>
on <relation name/view name>
to <user/user list>

grant update (budget) on department to Amit, Satoshi;
```

可以先创建角色(role)然后给用户授予角色。

4 形式化关系查询语言

4.1 关系代数

选择	σ	挑选出符合一定性质的元组 $\sigma_{\text{Sub}=\text{"Phy"} \wedge \text{age} > 30}(\text{teachers})$
投影	Π	只选出对应属性 $\Pi_{\text{ID}, \text{name}, \text{salary}}(\text{teachers})$
笛卡尔积	\times	将两个关系整合（简单并置，需要进一步筛选）
自然连接	$r \bowtie s = \Pi_{R \cup S}(\sigma_{r.A_1=s.A_1 \wedge \dots \wedge r.A_n=s.A_n}(r \times s))$	
θ 连接	$r \bowtie_{\theta} s = \sigma_{\theta}(r \times s)$	
外连接	$\bowtie, \bowtie_{\text{L}}, \bowtie_{\text{R}}, \bowtie_{\text{F}}$	
并集	\cup	数目应相同，属性可兼容
交集	\cap	
差集	$-$	
赋值	\leftarrow	
重命名	$\rho_x(E)$	给 E 的返回值赋名为 x

扩展的关系代数运算：

- 广义投影： $\Pi_{F_1, F_2, \dots, F_n}(E)$ ，其中 F_i 中每一个都是涉及常量及 E 的模式中属性的算术表达式
- 聚类： \mathcal{G} ，如count, min, max，考虑group by可以写成下列形式

$\text{dept_name} \mathcal{G}_{\text{average}(\text{salary})}(\text{instructor})$

定义 2 (聚集运算). 聚集运算 \mathcal{G} 的通常形式如下:

$$G_1, G_2, \dots, G_n \mathcal{G}_{F_1(A_1), \dots, F_n(A_n)}(E)$$

其中 E 是任意关系代数表达式, G_1, \dots, G_n 是用于分组的一系列属性; 每个 F_i 是一个聚集函数, 每个 A_i 是一个属性名。 E 结果中的元组将会以如下方式分为若干组:

- 同组中所有元组在 G_1, \dots, G_n 上的取值相同
- 不同组中的元组在 G_1, \dots, G_n 上的取值不同

4.2 其他关系演算

4.2.1 元组关系演算

元组关系演算是非过程化的查询语言 (有点像声明式语言), 它只描述所需信息, 而不给出获得该信息的具体过程。

$$t \in \text{instructor} \wedge \exists s \in \text{department}(t[\text{dept_name}] = s[\text{dept_name}])$$

前者 t 为自由变量, 后者 s 为受限变量。

4.2.2 域关系演算

$$\{ \langle x_1, x_2, \dots, x_n \rangle \mid P(x_1, x_2, \dots, x_n) \}$$

其中 $\langle x_1, \dots, x_n \rangle \in r$, 每一个 x_i 为域变量, r 为 n 个属性上的关系。

5 E-R模型

在设计一个数据库模式时, 要避免以下两个缺陷:

- 冗余: 可能导致不同关系表中的信息没有及时更新
- 不完整: 只有对应开课的实体而没有对应课程的实体

5.1 实体-关系模型

定义 3 (实体). 实体是现实世界中可区别于所有其他对象的一个事物或对象。每个实体有一组性质/属性(attribute), 其中一些性质的值可以唯一标识一个实体。实体集则是相同类型具有相同属性或性质的一个实体集合。实体集是一个抽象概念, 而实体集的外延(extension)则是指属于实体集实体的实际集合。

大学中实际教师的集合构成了实体集instructor的外延。实体集不必互不相交, 如定义大学里所有人的实体集(person)。一个person实体可以是instructor实体, 可以是student实体, 可以既是instructor实体又是student实体, 也可以都不是。

定义 4 (联系). 联系(*relationship*)是指多个实体间的相互关联。联系集是相同类型联系的集合。

$$\{(e_1, e_2, \dots, e_n)\} \mid e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n$$

联系也可以具有描述性属性 (如下右图)。

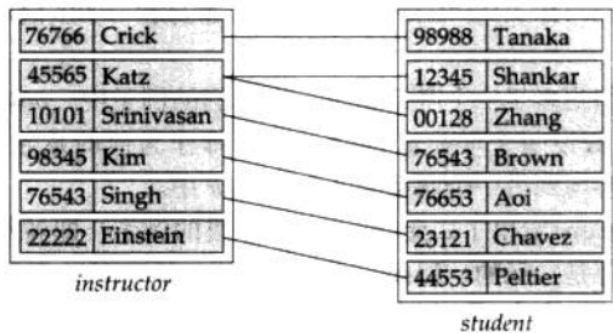


图 7-2 联系集 *advisor*

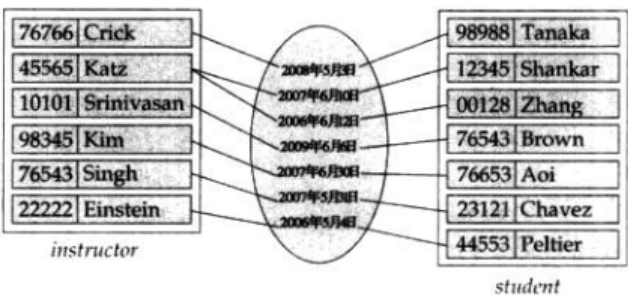


图 7-3 *date* 作为联系集 *advisor* 的属性

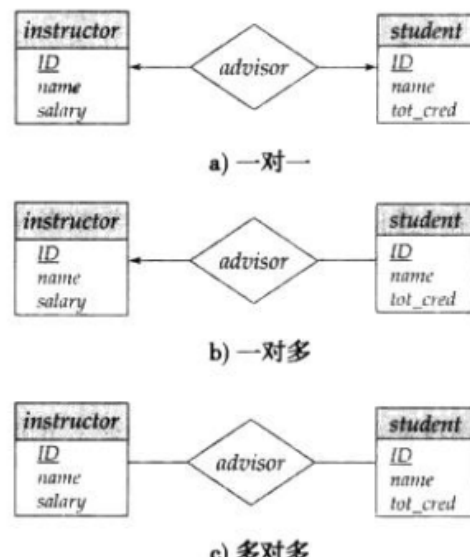
实体集之间的关联称为参与(*participation*), 若 E 中每个实体都参与到联系集 R 中的至少一个联系中, 则称参与是全部的(*total*)的, 否则是部分的(*partial*)。 E - R 模式中的联系实例则为具体命名实体间的关联。实体在联系中扮演的功能称为实体的角色(*role*)。参与联系集的实体集数目称为联系集的度(*degree*), 二元联系集的度为 2, 三元联系集度为 3。

定义 5 (属性). 每个属性都有一个可取值的集合, 称为该属性的域(*domain*), 或值集(*value set*)。属性可以分为简单和复合(*composite*)属性, 复合属性可以再划分为更小的部分, 如名字分为名和姓。也可能是单值或多值属性, 比如老师可以有多个电话号码, 这是多值属性。还有派生(*derived*)属性, 可以通过其他属性计算得出。

通常如果一个属性在两个实体集中出现, 且这两个实体集存在关联, 则该属性是冗余的, 需要被移除。

定义 6 (映射基数(mapping cardinality)/基数比率). 一个实体通过一个联系集能关联的实体个数, 包括了一对一、一对多(*many*)、多对一、多对多这几种情况。 *one* 代表至多一个, *many* 代表零个或多个。

- 一对一：我们从联系集 *advisor* 向实体集 *instructor* 和 *student* 各画一个箭头(见图 7-9a)。这表示一名教师可以指导至多一名学生，并且一名学生可以有至多一位导师。
- 一对多：我们从联系集 *advisor* 画一个箭头到实体集 *instructor*，以及一条线段到实体集 *student* (见图 7-9b)。这表示一名教师可以指导多名学生，但一名学生可以有至多一位导师。
- 多对一：我们从联系集 *advisor* 画一条线段到实体集 *instructor*，以及一个箭头到实体集 *student*。这表示一名教师可以指导至多一名学生，但一名学生可以有多位导师。
- 多对多：我们从联系集 *advisor* 向实体集 *instructor* 和 *student* 各画一条线段(见图 7-9c)。这表示一名教师可以指导多名学生，并且一名学生可以有多位导师。



也可以采用更为复杂的映射基数 $l..h$ 的形式表示，其中 l 表示最小映射基数， h 为最大映射基数。最小值为1代表这个实体集在该联系集中全部参与，最大值为1表示这个实体至多参与一个联系，最大值为*则没有限制。

定义 7 (强弱实体集). 没有足够属性形成主码的实体集称为弱实体集，有主码的实体集则称为强实体集。弱实体集必须与另一个称作标识(*identifying*)或属主(*owner*)实体集的实体集关联才有意义。弱实体集的分辨符(*discriminator*)/部分码是使得能够区分弱实体集中实体的方法，用虚下划线标明。

E-R图，菱形代表联系，双横线代表全部参与。

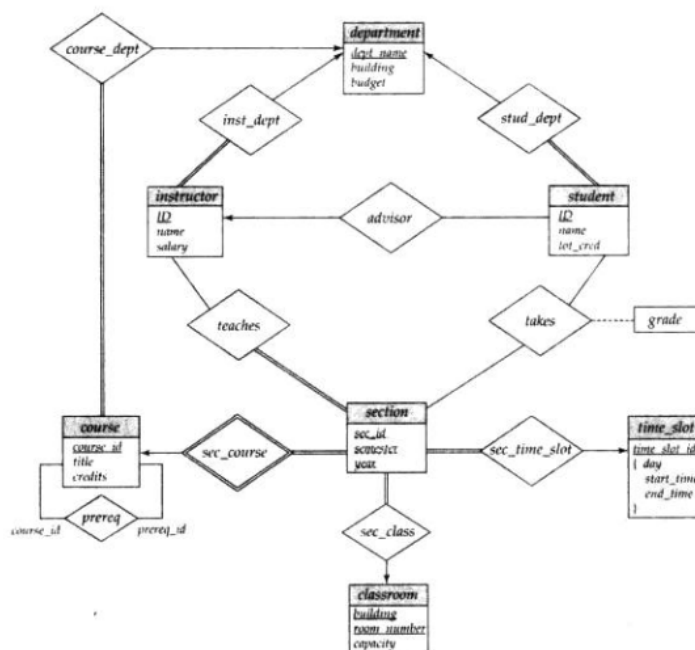


图 7-15 大学的 E-R 图

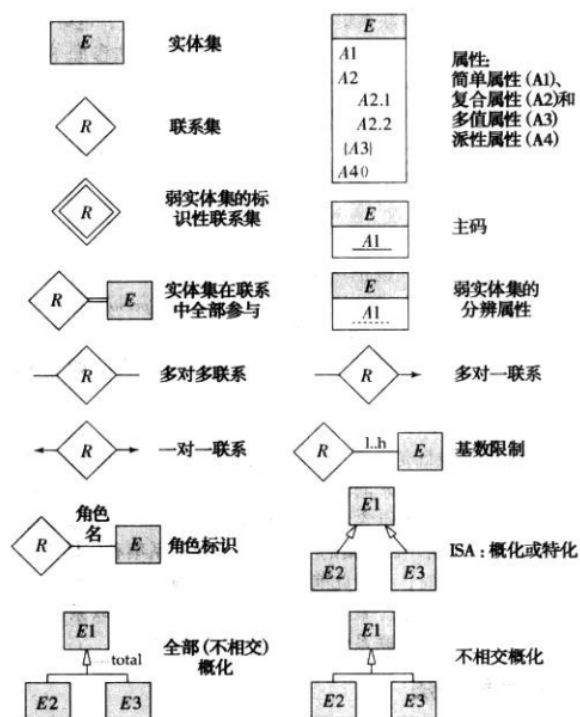


图 7-24 E-R 图表示法中使用的符号

一个常见的错误是将一个实体集的主码作为另一个实体集的属性（关系被隐藏了），而不是使用联系。

5.2 扩展的E-R属性

定义 8 (特化和泛化). 在实体集内部进行分组的过程称为特化(*specialization*), 如将实体集 *person* 划分为 *employee* 和 *student*. 概化(*generalization*)则是自底向上的反过程。高层与低层实体集也可被分别称为超类和子类, 同样有继承(*inheritance*)的特性。如果一个实体集作为低层实体集参与到多个联系中, 则称这个实体集具有多继承, 且产生的结构称为格(*lattice*)。

定义 9 (聚集). 聚集是一种抽象, 其中联系集和跟它们相关的实体集一起被看作高层实体集, 并且可以参与联系。

6 关系型数据库设计

数据库设计的范式(normal form)

- 第一范式：原子性，即所有元素都是不可分的单元
-

定义 10 (函数依赖(functional dependency)). 设 R 为关系模式, $\alpha \subset R, \beta \subset R$, 函数依赖 $\alpha \rightarrow \beta$ 在 R 上满足, 当且仅当对于任意合法的关系 $r(R)$, 任何两个关于 r 的数对 t_1 和 t_2 , 如果满足属性 α , 那么它们也满足属性 β , 即

$$t_1[\alpha] = t_2[\alpha] \implies t_1[\beta] = t_2[\beta]$$

实际上就是函数单射的概念。

考虑下例 $r(A, B)$ 的关系

A	B
1	4
1	5
3	7

关系 $A \rightarrow B$ 不成立，但关系 $B \rightarrow A$ 成立。

定义 11 (多值依赖(multivalued dependency)). 设 R 为关系模式, $\alpha \subset R, \beta \subset R$, 多值依赖在 R 上满足 $\alpha \twoheadrightarrow \beta$, 当且仅当对于所有数对 t_1 和 t_2 , 使得 $t_1[\alpha] = t_2[\alpha]$, 都存在数对 t_3 和 t_4 使得

$$\begin{aligned}t_1[\alpha] &= t_2[\alpha] = t_3[\alpha] = t_4[\alpha] \\t_3[\beta] &= t_1[\beta] \\t_3[R - \beta] &= t_2[R - \beta] \\t_4[\beta] &= t_2[\beta] \\t_4[R - \beta] &= t_1[R - \beta]\end{aligned}$$

简而言之, 设 (x, y, z) 有对应着属性 $\alpha, \beta, R - \alpha - \beta$, 则对于任意 R 中的元素 (a, b, c) 和 (a, d, e) , 都有 (a, b, e) 和 (a, d, c) 在 R 中。

7 事务

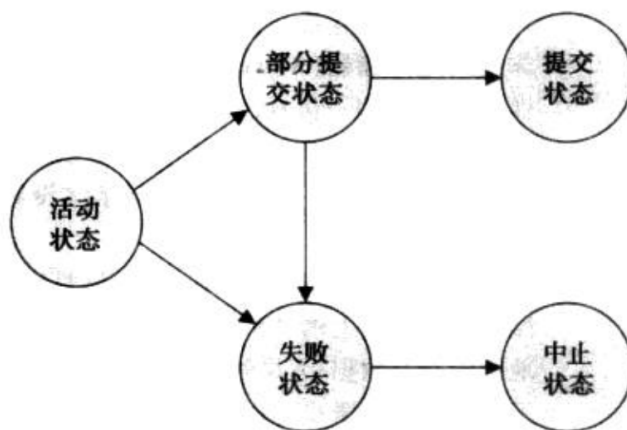
定义 12 (事务(transaction)). 构成单一逻辑工作单元的操作集合称为事务。即使有故障, 数据库系统也必须保证数据库的正确执行——要么执行整个事务, 要么属于该事务的操作一个也不执行。

事务具有以下的基本性质:

- 原子性: 要么执行完, 要么不执行, 不能执行到一半。
- 一致性: 除了基本的数据完整性约束, 还有更多的一致性约束。
- 隔离性: 每个事务都察觉不到系统中有其他事务在并发执行, 一定是完成一个再进行下一个。
- 持久性: 一个事务成功完成对数据库的改变是永久的, 即使出现系统故障。

事务的基本状态:

- 活动的(active): 初始状态
- 部分提交的(partially committed): 最后一条语句执行后
- 失败的(failed): 执行出错
- 中止的(aborted): 事务回滚且数据库已恢复到事务开始执行前
- 提交的(committed): 成功完成



定义 13 (冲突). 当 I 和 J 是不同事务在相同数据项上的操作，并且其中至少有一个是write指令时，则称 I 与 J 是冲突的。

定义 14 (冲突等价(conflict equivalent)). 如果调度 S 可以经过一系列非冲突指令交换转换为 S' ，则称 S 和 S' 是冲突等价的。若 S' 为串行调度，则 S 为冲突可串行化的(*conflict serializable*)。

可以通过构造一个优先图(precedence graph)来判断是否冲突可串行化。如果无环（用环检测算法），则冲突可串行化，可以通过拓扑排序得到串行化顺序。