数据库一天从入门到及格（上）

“末日时在做什么？有没有空？可以来学会儿数据库吗？”

Author:Yingluosanqian

by 陈一涵

本文帮助梳理数据库的知识点，有些叙述可能不太准确或者不太详细，建议以书中定义为准，比如各种范式，外键约束等。

前言

- 本文仅作理论分析，具体例子建议自行搜索。（懒得写）
- 个人认为，这门课程的部分很抽象很繁琐的内容，应当在实践（具体例子）中，认识问题，体会解决问题的思想，然后再去看定义，才能够明白为啥定义那么抽象得离谱。

第二章 关系数据类型

2.2 关系模型基础

2.2.0 前言

- 没啥好说的，很好理解的概念。
- 前置芝士：关系，是一个二维表。

2.2.1 属性

关系的列命名为属性（attribute）。

2.2.2 模式

关系名与其属性集合的组合成为这个关系的模式。

例如： Movies（title，year，length，genre）

2.2.3 元组

关系中除含有属性名所在行意外的其他行称作元组。

每一个元组都有一个分类对应于关系的每个属性。

2.2.4 域

域：用我的话来说，是指特定类型的属性的的取值范围。

2.2.5 关系的等价描述

关系是元组的集合，而不是元组的列表

2.2.6 关系实例

一个给定关系中的元组的集合叫做关系的实例。

2.2.7 关系上的键

键由关系的一组属性集组成，通过定义键可以保证关系实例上的任何两个元组的值在定义键的属性集上取值不同。

2.3 在SQL中定义关系模式

SQL由两方面的内容：

1. 用于定义数据库模式的数据定义子语言。
2. 用于查询和更新数据库的数据操纵语言。

2.3.1 SQL中的关系

1. 存储的关系，成为表（table）
2. 视图（view）
3. 临时表

2.3.2 数据类型

- 可变长度或固定长度的字符串。CHAR(n)表示最大为n个字符的固定长度字符串。
- 固定或可变长度的位串BIT(n)。
- BOOLEAN类型。它的取值可以是**TRUE、FALSE、UNKNOWN**。
- INT和INTEGER。
- FLOAT和REAL。
- DATA和TIME。

2.3.3 简单的表的定义

最简单的关系模式的定义形式是由保留字**CREATE TABLE**后面跟着关系名以及括起来的由属性名和类型组成的列表。

例如：

```
CREATE TABLE movies(
    title   CHAR(100),
    year    INT,
    length  INT,
    genre   CHAR(10),
    studioName  CHAR(30),
    producerC#  INT
);
```

2.3.4 修改关系模式

删除某个关系R，可以使用如下的SQL语句**DROP**：

```
DROP TABLE R;
```

增加某个属性，可以使用如下语句**ADD**：

```
ALTER TABLE MovieStar ADD phone CHAR(16);
```

关于**ALTER**语句：

```
ALTER TABLE MovieStar DROP birthdate;
```

2.3.5 默认值

为创建的属性设定默认值**DEFAUL**：

```
gender CHAR(1) DEFAULT '?'
birthdate DATE DEFAULT DATE '0000-00-00'
ALTER TABLE MovieStar ADD phone CHAR(16) DEFAULT 'unlisted'
```

2.3.6 键的声明

有两种方法

- **PRIMARY KEY**
- **UNIQUE**

```
CREATE TABLE MovieStar(
    name    CHAR(30)    PRIMARY KEY,
    address VARCHAR(55),
    gender CHAR(1),
    birthdate DATE
)

CREATE TABLE MovieStar(
    name    CHAR(30),
    address VARCHAR(55),
    gender CHAR(1),
    birthdate DATE,
    PRIMARY KEY (name,address)
)
```

2.4 代数查询语言

2.4.0 前言

- 书上关于关系代数的运算的描述⑧太星，建议参考https://www.icourse163.org/course/BNU-1002842007的第二章节。

2.4.1 为什么需要一种专门的查询语言

???这不是我这个挂科人该思考的问题

2.4.2 什么是代数

关系代数是一门代数，它的原子操作数是：

1. 代表关系的变量。
2. 代表优先关系的常量。

2.4.3 关系代数概述

传统关系代数的操作主要有以下四类：

1. 通常的关系操作：并，交，差。
2. 除去某些行或者列的操作。“选择”是消除某些行的操作，而“投影”是消除某些列的操作。
3. 组合两个关系元组的操作。包括“笛卡尔积运算”，以及很多“连接”操作。
4. “重命名”操作。不影响关系中的元组，但是它改变了关系模式，即属性的名称或是关系本身的名称被改变。

人们一般把关系代数的表达式称为查询。

2.4.4 关系上的操作

并：$R \bigcup S$
交：$R \bigcap S$
差：$R - S$

进行并交差操作时，需要满足：

1. R和S必须是具有相同的属性的表，同时，R和S各个属性的类型也必须匹配。
2. 在做相应的集合操作之前，R和S的列必须经过排序，这样保证它的属性序对于两个关系来说完全相同。

2.4.5 投影

投影（projection）操作用来从关系R生成一个新的关系，这个关系只包含原来关系R中的部分列。

表达式$\pi_{A_1,A_2 \ldots A_n}(R)$的值是这样的一个关系：它只包含关系R属性$A_1, A_2 \ldots A_n$所代表的列。

重要的事情说三遍，表达式的值是一个关系！
重要的事情说三遍，表达式的值是一个关系！
重要的事情说三遍，表达式的值是一个关系！

例如： $\pi_{title,year,length}(Movie)$

2.4.6 选择

选择（selection）操作符应用到关系R上时，产生一个关系R的元组的子集合。**结果关系的元组必须满足某个涉及R中属性的条件C，这个操作被表示为$\sigma_C(R)$。**

例如： $\sigma_{length \geq 100 \ \ AND \ \ studioName='Fox'}(Movie)$

2.4.7 笛卡尔积

关系R和S笛卡尔积是一个有序对的集合，有序对的第一个元素是关系R中的任一个元组，第二个元素是关系R中的任一个元组，表示为$R \times S$

PS： 学过离散的同学应该很清楚什么是笛卡尔积。
例如： $A = \{1, 2\}$ $B = \{a, b, c\}$，则

$$A \times B = \{(1, a), (1, b), (1, c), (2, a), (2, b), (2, c)\}$$

2.4.8 自然连接（natural join）

关系R和S的自然连接表示为$R \infty S$。此操作仅仅把R和S模式中有某共同属性，且此属性有相同的值的元组配对。

例如： https://www.icourse163.org/learn/BNU-1002842007?tid=1461106442#/learn/content?type=detail&id=1237413037&sm=1 2分30秒图中红框部分。（想看的例子这里都有，不再重复放了）

2.4.9 θ连接

关系R和S满足条件C的θ连接可以这样用符号来表示：$R \infty_C S$。这个操作的结果是这样构造的：

1. 先得到S和R的笛卡尔积
2. 在得到的关系中寻找满足条件C的元组。

（如有必要，在崇明的属性前面加上“R.”或“S.”。）

例如： $U \infty_{A<D \ \ AND \ \ U.B \neq V.B} V$

2.4.10 组合操作构成查询

例如： （以下二者等价）

$\pi_{title,year}(\sigma_{length>=100}(Movies) \bigcap \sigma_{studioName='Fox'}(Movies))$

$\pi_{title,year}(\sigma_{length>=100 \ \ AND \ \ studioName='Fox'}(Movies))$

2.4.11 命名与重命名

算符$\rho_s(A_1, A_2 \dots A_n)(R)$表示对关系R重新命名。重命名后的关系与关系R有完全相同的元组，只不过关系的名字变成了S。另外，S关系的各个属性分别命名为$A_1, A_2 \dots A_n$，按从左到右的顺序排列。如果只是想把关系R的名字变成S，并不改变其中属性的名字，就可以简单地使用$\rho_s(R)$

2.4.12 操作之间的联系

一些操作可以用其他代数操作来表达，例如

$$R \bigcap S = R - (R - S)$$
$$R \infty_C S = \sigma_C(R \times S)$$

2.4.13 代数表达式的线性符号

emmm我也不懂也不想懂.....

2.5 关系上的约束

书上是这么说的：

- 约束：关系模型对于存储在数据库中的数据具有的约束能力。
- 键约束：阿巴阿巴
- 引用完整性约束：要求一个关系属性列中出现的值也必须在同一个或不同的关系相应列中出现。
- （我怎么感觉它主要是想说怎么用关系代数表达式表达这些约束？）

2.5.1 作为约束语言的关系代数

用关系代数表示约束有如下两种方法：

1. 如果R是关系代数表达式，那么$R = \emptyset$表示“R中的值必须为空”的约束，与“R中没有元组”等价。
2. 如果R和S是关系代数表达式，那么$R \subseteq S$表示“任何在R中出现的元组都必须在S中出现”的约束。

例子：
$R = \emptyset \Leftrightarrow R \subseteq \emptyset$
$R \subseteq S \Leftrightarrow R - S = \emptyset$

2.5.2 引用完整性约束

引用完整性约束是一种普通的约束。它规定在某个上下文中出现的值也必须在另外一个相关的上下文中出现。举例来说，在Movies数据库中，关系StarsIn的一个元组的starName分量的值为p，那么，人们希望值p作为某个演员的名字也出现在关系MovieStar里。如果关系MovieStar里没有该值，则有理由怀疑“p”是否真的是一个演员。
概括来讲，如果关系R中的某个元组的属性分量（A）的值为v，那么按照设计意图，人们期望v也是另一个关系S的某个元组的一个相应的属性分量（B）的值。用关系代数将引用完整性表述为$\pi_A(R) \subseteq \pi_B(S)$。

2.5.3 键约束

直接上例子，{name}是MovieStar的键，键约束可表述如下：

$$\sigma_{MS1.name=MS2.name AND MS1.address \neq MS2.address}(MS1 \times MS2) = \emptyset$$

其中MS1和MS2分别是改名后的关系名。

2.5.4 其他约束举例

书P34页自己去看？

第三章 关系数据库设计理论

3.1 函数依赖

3.1.0 前置芝士（注）

- emmm，前置芝士有“关系”、“属性”、“元组”的概念。
- 本章常涉及“集合”的概念，集合的思想贯穿全篇，作为基础，如果对集合的理解不深入或对集合的应用不熟练，学起来难以深入到抽象的知识，需要借助更多的直观的理解。
- 在本章内容中，经常同时用到“属性”、“关系”、“函数依赖”这些概念，如果没有搞清楚这些定义将看得一头雾水。
- 学习中遇到的暂时不理解的抽象芝士，可以先对之进行直观上的理解，等熟练后再进行抽象学习，并从抽象知识中加深体会。

3.1.1 函数依赖的定义

关系R上的函数依赖（functional dependency，FD）是指 **“如果R的两个元组在属性$A_1, A_2, \ldots, A_n$上一致，那么它们必定在其他属性$B_1, B_2, \ldots, B_m$上也一致。”** 该函数依赖形式地记为

$$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$$

并称为“$A_1, A_2, \ldots, A_n$函数决定于$B_1, B_2, \ldots, B_m$”。

3.1.2 关系的键

如果下列条件满足，就认为一个或多个属性集多个属性集$\{A_1, A_2, \ldots, A_n\}$是关系R的键。

1. **这些属性函数决定关系的所有其他属性。** 也可以说，关系R不可能存在两个不同的元组，他们具有相同的$A_1, A_2, \ldots, A_n$值。
2. **在$A_1, A_2, \ldots, A_n$的子集中，没有一个能函数决定R的所有其他属性。** 也就是说，键必须是最小的。

PS： 当键只包含一个单独的属性A时，称A（而不是$\{A\}$）是键。
PS： 有时一个关系可能会有多个键。此时，通常指定其中一个为主键。

3.1.3 超键

一个包含键的属性集就叫做超键（superkey），它是“键的超集”的简写。 因此，每个键都是超键。然而，某些超键不是键。注意：每个超键都满足键的第一个条件：它决定了关系中所有其他属性。但超键不需要满足第二个条件：最小化。

3.2 函数依赖的规则

3.2.1 函数依赖的推导

在不改变关系合法实例集合的前提下，FD可以有多种不同的描述方法：

- 对于FD集合S和T而言，若关系实例集合满足S与其满足T的情况完全一样，就认为S和T等价（equivalent）。
- 更普遍的情况是，若满足T中的所有FD的每个关系实例也满足S中的所有FD，则认为S是从T中推断（follow）而来的。

3.2.2 分解/结合规则

注： 在了解分解/结合规则之前，最好先了解**Armstrong公理**，即

自反律：若属性集Y 包含于属性集X，属性集X 包含于U，则X→Y 在R 上成立。（此处X→Y是平凡函数依赖）
增广律：若X→Y 在R 上成立，且属性集Z 包含于属性集U，则XZ→YZ 在R 上成立。
传递律：若X→Y 和 Y→Z在R 上成立，则X →Z 在R 上成立。

书中的的分解规则和结合规则其实是Armstrong公理的部分推论（证明略），还有一个推论是**伪传递律**，即

$$若A_i \to A_j，\ A_jA_k \to A_l，则A_iA_k \to A_l$$

接下来正式介绍书中提到的**分解/传递规则**。

- 可以用一个FD的集合$A_1A_2\cdots A_n \to B_i(i=1,2,\ldots,m)$替换FD$A_1A_2\cdots A_n \to B_1B_2\cdots B_m$。这种转化成为**分解规则（splitting rule)**
- 可以用一个FD的集合$A_1A_2\cdots A_n \to B_1B_2\cdots B_m$替换FD$A_1A_2\cdots A_n \to B_i(i=1,2,\ldots,m)$。这种转化成为**组合规则(combining rule)**

3.2.3 平凡函数依赖

如果一个关系上的一个约束对所有的关系实例都成立，且于其他约束无关，则称其为**平凡的（trivial）**。当其给定FD时，能够很容易地判断一个FD是否是平凡的。平凡FD是这样也一类FD：$A_1A_2\cdots A_n \to B_1B_2\cdots B_m$，其中$\{B_1B_2\cdots B_m\} \subseteq \{A_1A_2\cdots A_n\}$。**也就是说，平凡FD的右边是左边的子集。**

例如： $title\ \ year \ \ \to \ \ title$

- FD$A_1A_2\cdots A_n \to B_1B_2\cdots B_m$等价于

  $$A_1A_2\cdots A_n \to C_1C_2\cdots C_k$$

  其中，C是集合B中而不是集合A中的属性，这个规则被称为**平凡的依赖规则（trivial-dependency rule）**。

3.2.4 计算属性的闭包

闭包的定义： 假设$A_1, A_2, \ldots, A_n$是属性集合，**S是FD的集合。则S集合下的属性集合$\{A_1, A_2, \ldots, A_n\}$的闭包（closure）是满足下面条件的属性集合B**，即使得每一个满足S中所有FD的关系，也同样满足$A_1, A_2, \ldots, A_n \rightarrow B$。（吐槽：这说的什么**玩意儿）。也就是说，$A_1, A_2, \ldots, A_n \rightarrow B$能从S的FD中推断出来。属性集合$\{A_1, A_2, \ldots, A_n\}$的闭包记为$\{A_1, A_2, \ldots, A_n\}^+$.

注：用作者的话说，根据S中的规则，用属性集合A中现有的元素拓展新的元素，直到无法再拓展新的元素，即得到A^+

属性集合的闭包算法： 阿巴阿巴

```
(^*(^^*&!@&*%@&#^(@#^)))
```

（有兴趣查阅书中P42）简单来说就是刚刚用我的话说的，一直利用规则拓展，直到无法拓展，事实上看一个例子就懂了，但是我懒得写例子。

3.2.5 闭包算法为何有效

证明略。

3.2.6 传递规则

即**Armstrong公理**中的传递律

3.2.7 函数依赖的闭包集合

基本集：如果给定一个FD集合S，则任何和S等价的FD集合都被称为S的基本集（basis）。

最小化基本集： 满足下面三个条件的基本集B被称为关系的最小化基本集（minimal basis）。

1. B中所有的FD的右边均为单一属性。（注：容易理解）
2. 从B中删除任何一个FD后，该集合不再是基本集。（注：即，可以被由其他FD推导得到的FD应被去除）
3. 对于B中的任何一个FD，如果从其左边删除一个或多个属性，B将不再是基本集。（注：即，去除所有FD左边属性的冗余）

3.2.8 投影函数依赖

假设有一个含有FD集合S的关系R，通过计算$R_1 = \pi_L(R)$得到对其部分属性的投影。那么R_1中有哪些FD成立?

FD集S的投影是所有满足下面条件的FD集合：

1. 从S推断而来。
2. 只包含R_1的属性。

算法 书P46（简单来说就是一个从闭包出发的暴力算法，但书P46页的例3.13给出了一些剪枝技巧）

3.2.X 我的心得

- 本章的闭包算法是一个很容易理解的算法，很多需要问题都可以使用闭包算法时，都可以得到严谨、规范的解决，但缺点是闭包算法过于繁琐。

3.3 关系数据库模式设计

3.3.0 前言

- 教材上这章节的知识点分布实在离谱，对于入门可能逻辑上比较顺畅，但是出于复习实在是太乱了。于是我重新分了章节。先统一介绍所有范式，再介绍算法，再介绍分解及其优劣。
- 这里有很多很繁琐的定义，如果想要彻底搞清楚，最好在了解为什么它被提出之后，再去看具体的定义，就会明白为什么这么定义。而不是先看定义然后一脸懵逼。当然，出于一天从入门到及格，对于范式而言，背一背定义是极好的。

3.3.1 1NF

如果关系模式S的每个关系的每个属性值都是不可分的原子值，称S是第一范式的模式。

注： 关系数据库只研究满足1NF的关系。简单来说，绝大多数我们做过的题目中，某个关系模式都至少满足1NF。

3.3.2 2NF

若关系模式S满足2NF，则对于任意非平凡函数依赖$A_1A_2\cdots A_n \to B$，若B不是键属性，则$A_1A_2\cdots A_n$也不是某个键的真子集。

注： 我不确定这个是不是充要条件，但是老师上课的时候是这么把它当充要条件用的，因此，我暂且凭直觉把它作为充要条件。并且，通过这个条件，我们至少得到了一个判定某个关系模式S是否满足2NF的方法：**如果存在$A_1A_2\cdots A_n \to B$，其中B不是键属性，然鹅$A_1A_2\cdots A_n$却是某个键的真子集，那么它就不满足2NF。其他任意情况均满足2NF。**

注： 给出一个2NF的定义（相关概念自行查找，可以不需要了解）：若关系模式$S \in 1NF$，且每一个非主属性都不部分依赖于S的任何候选键，则$S \in 2NF$。

3.3.3 3NF

关系R满足3NF，如果它满足：

- 只要$A_1A_2\cdots A_n \rightarrow B_1B_2\cdots B_m$是非平凡FD，那么或者$\{A_1A_2\cdots A_n\}$是超键，或者每个属于$\{B_1B_2\cdots B_m\}$但不属于A的属性都是某个键的成员（所属的键可以不同）

换一种直观的表述就是："对于每个非平凡FD，或者左边是超键，或者右边仅由主属性构成"。

3.3.4 BCNF

关系R属于BCNF当且仅当：如果R中非平凡FD$A_1A_2\cdots A_n \rightarrow B_1B_2\cdots B_m$成立，则$A_1A_2\cdots A_n$是关系R的超键。

注： 对比3NF，容易发现BCNF相比3NF，不容许"右边仅有主属性构成"。

注： 书P50通过枚举所有情况，进行分类讨论证明了，任意一个二元关系都属于BCNF。（这意味着任意一个关系模式都可以通过不断分解，从而满足BCNF）

3.3.5 多值依赖（multivalued dependency）

参考链接1：https://blog.csdn.net/smstong/article/details/5599490?ops_request_misc=%257B%2522request%255Fid%2522%253A%252216093183091678027704622 1%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fblog.%2522%257D&request_id=160931830916780277046221&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~blog~first_rank_v2~rank_v29-2-5599490.pc_v2_rank_blog_default&utm_term=多值依赖

参考链接1：https://blog.csdn.net/smstong/article/details/5599609?ops_request_misc=%257B%2522request%255Fid%2522%253A%252216093183091678027704622 1%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fblog.%2522%257D&request_id=160931830916780277046221&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~blog~first_rank_v2~rank_v29-1-5599609.pc_v2_rank_blog_default&utm_term=多值依赖

在叙述定义之前，希望读者明白一件事情，这个多值依赖它的抽象定义很迷惑，建议先搞懂多值依赖它想说的是什么，再去看定义的时候思考为什么这么定义？这样定义可以精确地描述你脑子里想的那个"多值依赖"吗?

书上的定义：

- 在关系R中，当给定某个属性集合的值时，存在另外一组属性集合，该组属性的值于关系中所有其他属性的值独立。
- 精确地说，若给定R中属于A的各属性的值，存在一个属性集B，其中属性的值独立于R中既不属于A也不属于B的属性集合的值，则称MVD

$$A_1A_2\cdots A_n \rightarrow\rightarrow B_1B_2\cdots B_m$$

在R中成立。

- 更准确的说法是，若要MVD成立，则R中每个在所有A属性上一致的元组对t和u，能在R中找到满足下列条件的元组v：
  1. 在A属性上的取值与t和u相同；
  2. 在B属性上的取值与t相同；
  3. 在R-A-B属性上的取值与u相同。

这个定义很离谱，我更喜欢下一个，从网上查找到的：

多值依赖的定义：

设R(U)是一个属性集合U上的一个关系模式，
X，Y，和Z是U的子集，并且Z=U-X-Y，
多值依赖X->->Y成立当且仅当对R的任一个关系r，
r在(X,Z)上的每个值对应一组Y的值，
这组值仅仅决定于X值而与Z值无关。

//下面的定义似乎与书中的定义不太一样啊，不过我打算以这个定义为准
若X->->Y，而Z=空集，则称X->->Y为平凡的多值依赖。
否则，称X->->Y为非平凡的多值依赖。

3.3.6 多值依赖的推导

1. 平凡MVD：如果$\{B_1B_2\cdots B_m\}\subseteq\{A_1A_2\cdots A_n\}$，则MVD
$$A_1A_2\cdots A_n\rightarrow\rightarrow B_1B_2\cdots B_m$$
在任何关系中成立。
2. 传递规则：如果关系中存在$A_1,A_2,\ldots,A_n\rightarrow\rightarrow B_1,B_2,\ldots,B_m$和$B_1,B_2,\ldots,B_m\rightarrow\rightarrow C_1,C_2,\ldots,C_k$，则
$$A_1,A_2,\ldots,A_n\rightarrow\rightarrow C_1,C_2,\ldots,C_k$$
也成立。C的任何属于A的属性要从右边除去。
3. FD升级规则：每个FD都是MVD。
4. 互补规则：若关系R上存在$A_1,A_2,\ldots,A_n\rightarrow\rightarrow B_1,B_2,\ldots,B_m$，则R上也存在$A_1,A_2,\ldots,A_n\rightarrow\rightarrow C_1,C_2,\ldots,C_k$其中C=R-A-B。
5. 附加平凡MVD：$A_1,A_2,\ldots,A_n\rightarrow\rightarrow B_1,B_2,\ldots,B_m$成立，当R=A+B。

3.3.7 4NF

若对于R中的每个非平凡MVD$A_1,A_2,\ldots,A_n\rightarrow\rightarrow B_1,B_2,\ldots,B_m$，$\{A_1A_2\cdots A_n\}$都是超键，则R属于第四范式。

PS： 这个和BCNF很像，把MVD换成FD毫无违和感。

3.3.8 异常

1. 冗余（redundancy）.信息没有必要地在多个元组中重复。
2. 更新异常（update anomaly）。可能修改了某个元组的信息，但是没有改变其他元组的相同信息。
3. 删除异常（deletion anomaly）。如果一个值集变成空集，就可能带来丢失信息的副作用。

这部分的内容建议参考书P48页。

3.3.9 分解关系

一般用分解关系的方法来消除异常。关系R的分解涉及分离R的属性，以构造两个新的关系模式。

给定一个$R(A_1, A_2, \ldots, A_n)$，把它分解为关系$S(B_1, B_2, \ldots, B_m)$和$T(C_1, C_2, \ldots, C_k)$，并且满足：

1. $\{A_1, A_2, \ldots, A_n\} = \{B_1, B_2, \ldots, B_m\} \bigcup \{C_1, C_2, \ldots, C_k\}$
2. $S = \pi_{B_1, B_2, \ldots, B_m}(R)$。
3. $T = \pi_{C_1, C_2, \ldots, C_k}(R)$。

3.3.10 分解的优劣

一个分解具有以下三个性质：

1. 消除异常。
2. 信息的可恢复
3. 依赖的保持

注： 第二条和第三条性质不能只停留在字面意思上，建议自行结合例子弄清楚，我懒得写了。**以下内容将基于已经搞清楚了前文的前提进行叙述。**
注： 如果要求分解一定具有无损连接性，一定能达到BCNF；如果要求分解一定具有无损连接性和保持依赖性，一定能达到3NF。

3.3.11 无损连接的chase检验算法

建议参考书P57，结合例子理解。

个人理解： 直观上来看，造成“无法无损连接”的原因是是“某个不是键的东西”充当了连接的“媒介”，从而引起冗余（一个属性集不是键的话就可能有不同的元组在这个属性集上有同样的值，从而引起冗余）。因此，实际上只要检查连接的“媒介”是否都是键就可以了。因此，这个算法实际上是提供了这样一套的方法。而最终出现的一行“不带下标的字母”，本质上是找到了这样一种连接方式：连接的媒介都是键。

具体例子可参考：https://www.icourse163.org/learn/BNU-1002842007?tid=1461106442#/learn/content?type=detail&id=1237413093&cid=1257339134&replay=true第七章第

二节中8分22秒-9分53秒。

3.3.12 4NF分解算法

找到一个4NF违背式$X \rightarrow\rightarrow Y$，将属性集分解为$X \bigcup Y$和$X \bigcup \{A - X - Y\}$，递归处理这两个属性集。

PS： 建议联系BCNF的分解算法，理解记忆。

3.3.13 BCNF分解算法

找到一个BCNF违背式$X \rightarrow Y$，将属性集分解为X^+和$X \bigcup \{A - \{X^+\}\}$，递归处理这两个属性集。

P.S. 给出的BCNF的分解算法和4NF的分解算法其实是等价的。（如果把$X \rightarrow\rightarrow Y$改成$X \rightarrow Y$。

3.3.14 3NF分解算法

对于关系模式$S < A, D >$，

1. 找到D的最小化基本集，并将其左部相同的FD结合起来。
2. 每个函数依赖$\alpha \rightarrow \beta$构成一个模式$\alpha \bigcup \beta$。
3. 构成的模式集中，若每个模式都不包含键，则将任意一个键作为模式加入模式集。

PS： 这个算法简单多了吧。

3.3.X 总结

- 事实上，范式越高，异常越能被消除，写入越方便。但在读取时，由于需要合并不同的表，因此速度较慢。范式越低则相反。因此高范式适合以写为中心的情况，低范式适合以读为中心的情况，在实际应用时需要根据实际情况折中考虑。