

- Normal Forms and Schema Decomposition （范式及模式分解）
 - 规范化
 - 函数依赖
 - 码
 - 范式
 - 1NF
 - 2NF
 - 3NF
 - BCNF ?
 - 闭包，最小函数依赖集
 - 闭包
 - 最小函数依赖集 !
 - 模式分解 !
 - 模式分解的三个定义
 - 判断分解是否具有无损连接性（表格法） ?
 - 典型题型

Normal Forms and Schema Decomposition （范式及模式分解）

数据库系统 | 笔记整理（8）——关系数据理论 - 知乎 (zhihu.com)

规范化

函数依赖

函数依赖是关系模型中的一个基本概念，表示一种属性之间的依赖关系。如果对于关系中的任意两个元组，当它们在一组属性A上的值相同时，那么它们在另一组属性B上的值也必定相同，则称属性B函数依赖于属性A，记作 $A \rightarrow B$ 。

例子： 在学生信息表中，学生编号 (StudentID) 唯一确定了学生的姓名 (Name)，则可以表示为 $StudentID \rightarrow Name$ 。

- 非平凡函数依赖(Non-trivial Functional Dependency)

- 非平凡函数依赖指的是，在函数依赖 $A \rightarrow B$ 中， B 并不是 A 的子集。这种函数依赖描述了属性之间的实质性依赖关系。
- 例子： 在上面的例子中， $\text{StudentID} \rightarrow \text{Name}$ 就是一个非平凡函数依赖，因为 Name 不是 StudentID 的子集。
- 平凡函数依赖(Trivial Functional Dependency)
 - 平凡函数依赖指的是依赖关系 $A \rightarrow B$ 中， B 是 A 的子集。这种依赖是显而易见的，对数据库设计规范化没什么实质性帮助。
 - 例子： 在学生信息表中，假设有组合属性 $\{\text{StudentID}, \text{Name}\}$ ，则 $\{\text{StudentID}, \text{Name}\} \rightarrow \text{Name}$ 就是一个平凡函数依赖。
- 决定因素(Determinant)
 - 决定因素是指在函数依赖 $A \rightarrow B$ 中，能决定其他属性值的属性集 A 。
 - 例子： 在 $\text{StudentID} \rightarrow \text{Name}$ 中， StudentID 就是决定因素。
- 完全函数依赖(Full Functional Dependency)
 - 如果 $A \rightarrow B$ 是一个函数依赖，并且对于 A 的任何一个真子集 S ， $S \rightarrow B$ 不成立，则称 B 对 A 完全函数依赖。记作 $X \xrightarrow{F} Y$ 。
 - 例子： 设有关系 $R(\text{StudentID}, \text{CourseID}, \text{Grade})$ ，如果 $\{\text{StudentID}, \text{CourseID}\} \rightarrow \text{Grade}$ 是成立的，但是单独的 StudentID 或 CourseID 都不能确定 Grade ，则 Grade 对 $\{\text{StudentID}, \text{CourseID}\}$ 完全函数依赖。
- 部分函数依赖(Partial Functional Dependency)
 - 部分函数依赖是指在复合属性 $A \rightarrow B$ 的情况下，存在 A 的一个真子集 S 使得 $S \rightarrow B$ 也成立。记作 $X \xrightarrow{P} Y$ 。
 - 例子： 仍然使用关系 $R(\text{StudentID}, \text{CourseID}, \text{Grade})$ ，如果假设 $\{\text{StudentID}, \text{CourseID}\} \rightarrow \text{Grade}$ ，但是发现 $\text{StudentID} \rightarrow \text{Grade}$ 也成立（假设一个学生在所有课程中的成绩都相同），则称 Grade 对 $\{\text{StudentID}, \text{CourseID}\}$ 部分函数依赖。
- 传递函数依赖(Transitive Functional Dependency)
 - 传递函数依赖发生在三个属性 A 、 B 、 C 中，如果存在 $A \rightarrow B$ 和 $B \rightarrow C$ 的依赖关系，但是 C 不直接依赖于 A （ $A \rightarrow C$ 不成立），则称 C 对 A 传递函数依赖。
 - 例子： 设有关系 $R(\text{StudentID}, \text{MajorID}, \text{DepartmentName})$ ，如果 $\text{StudentID} \rightarrow \text{MajorID}$ 且 $\text{MajorID} \rightarrow \text{DepartmentName}$ 成立，但是 StudentID 到

DepartmentName没有直接的依赖关系，则称DepartmentName对StudentID传递函数依赖（通过MajorID）。

码

设 K 为 R 中的属性或属性组合。若 K 函数决定 U ，则 K 称为 R 的候选码（candidate key）

如果 U 部分函数依赖于 K ，即 $K \twoheadrightarrow U$ ，则 K 称为超码（surpkey）。候选码是最小的超码，即 K 的任意一个真子集都不是候选码。若关系模式 R 有多个候选码，则选定其中的一个为主码（primary key）。

- 包含在任何一个候选码中的属性，称为主属性（prime attribute）
- 不包含在任何一个候选码中的属性称为非主属性（nonprime attribute）或非码属性（non-key attribute）

最简单的情况，单个属性是码；最极端的情况，整个属性组是码，称为全码（all-key）。

Keys of a relation

Super-key:

- Is a set of attributes within a table(relation) whose values can be used to uniquely identify a row in the relation.

Candidate key:

- An attribute or minimal set of attributes, which uniquely identifies a row in a relation (a unique identifier).
- One of the candidate keys will become the primary key.
- Each no-key field is functionally dependent on every candidate key.

Primary key:

- Is a unique identifier
- It cannot contain null values
- One of the candidate keys will become the primary key.

范式

关系数据库中的关系是要满足一定要求的,满足不同程度要求的为不同范式

一个低一级范式的关系模式通过模式分解(schema decomposition)可以转换为若干个高一级范式的关系模式的集合,这种过程就叫规范化(normalization)。

1NF

- Every attribute value is atomic(atomic attributes cannot be divided into subparts)
(必要)

Example: In the following relation StuAddress is a non-atomic attribute and should be divided to smaller parts

Student(StudentID, StuDateOfBirth, StuAddress)

Student(StudentID, StuDateOfBirth, *StuUnitNumber,
StuStreet, StuSuburb, StuState*)

- No multivalued attributes(multivalued attributes can have more than one value at a time)
 - Multivalued attributes (多值属性): Attributes that can have more than one value at a time.

第一范式是数据库范式理论中的一个基础概念，它要求关系中的每一个属性都必须是不可分割的基本数据项。换句话说，数据的每一列都是不可分割的最小单元。有两个基本特性：

每个属性值都是原子的（原子属性不能被分解为子部分）（必要条件）例子：在下面的关系中，StuAddress是一个非原子属性，应被分解为更小的部分：Student（学生编号，学生出生日期，学生地址）我们可以根据第一范式将上面的表分解为：Student（学生编号，学生出生日期，单位号，街道，郊区，州）

没有多值属性（多值属性可以一次具有多个值）多值属性：一次可以有多个值的属性。这意味着，如果一个属性可能有多个值，那么根据第一范式，我们需要将该属性分解到其他表中，并在主表中使用引用。这样可以确保每个属性只有一个值。

2NF

[数据库范式（1NF 2NF 3NF BCNF）详解_如何判断1nf2nf3nf和bcnf-CSDN博客](#)

Second normal form: 2NF

1. 1NF PLUS every non-key attribute is fully functionally dependent on the ENTIRE primary key.

a. Every non-key attribute must be defined by the entire key, not by only part of the key

b. NO partial functional dependencies

Solution

1. Create new relation for each primary key (PK) and move non-key attributes that are only dependent on this PK.

2. Consider this PK as a foreign key (FK) in the original table(relation).

定义：若关系模式 $R \in 1NF$ ，且每一个非主属性完全函数依赖于任何一个候选码，则 $R \in 2NF$ 。

第二范式（**2NF**）在第一范式（**1NF**）的基础上增加了对关系模式的进一步要求，旨在消除部分函数依赖，从而减少数据冗余和更新异常。详细来说：

1. 完全函数依赖：在**2NF**中，每个非键属性必须完全函数依赖于整个主键。这意味着一个属性如果依赖于主键的一部分，那么这个关系模式就不满足第二范式。完全函数依赖意味着：
 - a. 每个非键属性都必须由整个主键定义，而不仅仅是主键的一部分。如果一个关系的主键是由多个属性组成的组合键，那么任何一个非键属性不应该只依赖于组合键的一部分，而应该依赖于整个组合键。
 - b. 没有部分函数依赖：部分函数依赖是指一个属性只依赖于组合键中的一部分，而不是全依赖于整个组合键。**2NF**要求去除这种依赖关系。
2. 解决办法 为了将关系模式转换到**2NF**，我们可以采取以下步骤：
 - a. 为每个组合主键创建新的关系模式，并将只依赖于该部分主键的非键属性移到这个新关系中。
 - b. 在原始表（关系）中考虑这个新创建的主键作为外键（**FK**）使用。

这样的处理达到两个目的：一方面，每个关系模式中的非键属性都直接依赖于它们所在的那个主键，符合**2NF**要求；另一方面，引入外键保持了不同关系模式之间的联系，维持了数据的完整性。

实例

假设我们有一个关系模式 ``OrderDetails` (OrderID, ProductID, Quantity, ProductPrice)``，其中 ``OrderID`` 和 ``ProductID`` 组成复合主键。如果 ``ProductPrice`` 只依赖于 ``ProductID`` 而不是整个主键，那么就存在部分函数依赖关系。

为了调整到**2NF**，我们可以这样操作：

1. 创建一个新的关系模式 ``Products` (ProductID, ProductPrice)``，将 ``ProductPrice`` 独立出来，因为它只依赖于 ``ProductID``。
2. 在原始的 ``OrderDetails`` 关系模式中保留 ``OrderID`` 和 ``ProductID`` 作为主键，``Quantity`` 作为非键属性，同时将 ``ProductID`` 作为外键引用新的 ``Products`` 关系模式。

这样一来，就消除了部分函数依赖，``OrderDetails`` 和 ``Products`` 两个关系模式都满足第二范式的要求。

假设我们有一个学校数据库中的关系模式 `ClassAssignments`，它记录了老师教授的课程和相关的班级

信息。以下是该关系模式的初步设计，它当前处于第一范式（1NF）：

ClassAssignments(TeacherID, CourseID, ClassName, TeacherName, CourseName)

其中，(TeacherID, CourseID) 一起形成了这个表的复合主键，因为一个老师可以教授多门课程，一门课程也可以由多位老师教授。每个教师和课程的组合都有一个指定的班级名称。

在这个设计中，有以下问题：

ClassName 完全函数依赖于复合主键 (TeacherID, CourseID)。

TeacherName 只部分函数依赖于TeacherID，而非整个复合主键。

CourseName 只部分函数依赖于CourseID，而非整个复合主键。

这表明 ClassAssignments 还未满足2NF。为了满足2NF，我们需要移除这些部分依赖性。

解决办法:我们可以将 ClassAssignments 分解为三个表来消除部分函数依赖，如下：

Teachers(TeacherID, TeacherName)

此表包含教师的信息。每个教师有一个唯一的 TeacherID 和对应的 TeacherName。

Courses(CourseID, CourseName)

此表包含课程信息。每门课程都有一个唯一的 CourseID 和对应的 CourseName。

ClassAssignments(TeacherID, CourseID, ClassName)

此表包含指定的课程分配信息，它使用 TeacherID 和 CourseID 作为复合主键连接老师和课程，同时 ClassName 完全依赖于这个复合主键。

在这种设计下，Teachers 和 Courses 表分别承担了解除 TeacherName 和 CourseName 的部分依赖性的任务。这样，ClassAssignments 表中只有完全函数依赖，满足了2NF的要求。

同时可以在 ClassAssignments 表中包含 TeacherID 和 CourseID 作为外键引用 Teachers 和 Courses 表，保持数据的引用完整性。

外键：我们除了 ClassName 这个普通列外，还定义了两个列 TeacherID 和 CourseID。然后，我们通过在两个列后追加 FOREIGN KEY 约束，指明这两个列为外键，它们分别引用 Teachers 表和 Courses 表的主键

3NF

[数据库设计的三范式超详细详解_数据库三范式-CSDN博客](#)

Third normal form: 3NF

- 2NF PLUS no transitive dependencies(functional dependencies on non-primary-key attributes)
- NOTE: this is called transitive, because the primary key is a determinant for another attribute, which in turn is a determinant for a third.

Solution

1. Non-key determinant with transitive dependencies go into a new table;
2. Non-key determinant becomes primary key in the new table, and
3. Stays as foreign key in the old table(relation)

第三范式（3NF）是在第二范式（2NF）的基础上对关系模式提出的进一步要求，目的是消除数据中的传递函数依赖，从而进一步减少数据的冗余性。明确来说：

1. 没有传递函数依赖 传递函数依赖是指当一个属性（A）依赖于另一个属性（B），且这个属性（B）依赖于主键，那么属性（A）就处于对主键的传递依赖的位置。
3NF要求一个关系模式中不应存在非主键属性对主键的传递函数依赖。注：被称

为传递的原因是，主键是另一属性的决定因素，而该属性又是第三个属性的决定因素。

2. 解决办法 为了将一个关系模式调整到第三范式，我们可以采取以下步骤： a. 将具有传递依赖的非主键决定因素移到新的表中。 b. 在新表中，这个非主键决定因素将成为主键。 c. 在旧表（关系）中，这个非主键决定因素将保持为外键。

通过这样的处理，我们既保留了数据的完整性，也消除了非主键属性对主键的传递函数依赖。

假设有一个关系模式 `Students(StudentID, StudentName, DepartmentID, DepartmentName)`，主键是 `StudentID`，`DepartmentName` 依赖于 `DepartmentID`（非主键），`DepartmentID` 依赖于 `StudentID`。这就形成了一个传递函数依赖。调整到第三范式，我们可以：创建一个新的关系模式 `Departments(DepartmentID, DepartmentName)`，将 `DepartmentName` 独立出来，因为它依赖于 `DepartmentID` 而非直接依赖于主键 `StudentID`。在原来的 `Students` 关系模式中，保留 `StudentID` 和 `StudentName` 两个属性，将 `DepartmentID` 作为外键，引用新的 `Departments` 关系模式。

BCNF ?

- A table is already in 3NF.
- Every determinant must be a **candidate** key

Simpler definition: a relation is not in BCNF if:

o Non-key -> part of key

o Part of key -> part of key

好像就是不能存在关键字段决定关键字段把

闭包，最小函数依赖集

闭包

Attribute closure (属性闭包)

- Attribute closer: if A is an attribute(or a combination of attributes), the set of attributes in relation R tat are functionally dependent on A is called attribute closure of A and it can be represented as A^+ .
- Steps to find the attribute closure of A
 1. Add A to the attribute closure set of A (A^+)
 2. Recursively add attributes which can be functionally determined from

attributes of the set A^+ until done.

属性闭包（Attribute Closure）是数据库系统中一个重要的概念。它主要用于确定一个给定的属性集合（可以是单个属性也可以是多个属性的组合）能够函数依赖获得的所有属性的集合。属性闭包对于理解数据库关系的键和函数依赖是非常有用的。

1. 属性闭包定义：如果 A 是一个属性或属性的组合，那么在关系 R 中，所有函数依赖于 A 的属性的集合称为 A 的属性闭包，表示为 A^+ 。简单来说，属性闭包 A^+ 是关系 R 中所有能由 A 直接或间接决定的属性的集合。
2. 找到属性闭包 A^+ 的步骤：以下是计算属性闭包的步骤：
a. 将 A 添加到其属性闭包集 A^+ 中。换句话说，属性闭包开始时包含该属性集本身。
b. 递归地将所有可以由 A^+ 集合中的属性函数确定的其他属性加入到集合 A^+ 中。每一次当你发现有某些属性是由 A^+ 中已有的属性函数依赖推导出的，就将这些属性添加到 A^+ 中。
c. 重复步骤 b 直到没有新的属性可以被添加到集合 A^+ 中为止。当再也没有新属性可以加入 A^+ 时，这个过程结束

例1, 设关系 $R(A, B, C, D, E, G)$ 有函数依赖集 $F=\{AB \rightarrow C, BC \rightarrow AD, D \rightarrow E, CG \rightarrow B\}$ ，求 AB 的闭包。

解：首先从 AB 出发，令 $X=\{A, B\}$ ，由于函数依赖 $AB \rightarrow C$ 左边的所有属性都在 X 中，所以可以把右边的 C 添加到 X 中，这时 $X=\{A, B, C\}$ 。

其次考虑函数依赖 $BC \rightarrow AD$ ，左边 B, C 均在 X 中，右边 D 不在 X 中，将其添加到 X 中，此时 $X=\{A, B, C, D\}$ 。

再考虑函数依赖 $D \rightarrow E$ ，同理可将 E 添加到 X 中，此时 $X=\{A, B, C, D, E\}$ 。

上述方法再不能向 X 中添加属性，所以得到 $\{A, B\}^+ = \{A, B, C, D, E\}$ 。

最小函数依赖集 !

补充(2): Prob: 求最小函数依赖集 F_{min} .

例: 已知关系模式 $R < U, F >$, $U = \{A, B, C, D, E, G\}$; $F = \{AB \rightarrow C, C \rightarrow A, CG \rightarrow BD, ACD \rightarrow B\}$, 请将 F 化为最小函数依赖集.

解: (1): 利用分解规则, 将所有的函数依赖变成右边都是单个属性的函数依赖:

$$F = \{AB \rightarrow C, C \rightarrow A, CG \rightarrow B, CG \rightarrow D, ACD \rightarrow B\}.$$

(2): 去掉 F 中多余的函数依赖.

①: 设 $AB \rightarrow C$ 为多余的, 则去掉 $AB \rightarrow C$ 得 $F_1 = \{C \rightarrow A, CG \rightarrow B, CG \rightarrow D, ACD \rightarrow B\}$

因为从 F_1 中找不到左部为 AB 或 AB 子集的函数依赖, 则 $(AB)_{F_1}^+ = AB$, 不包含 C ,
 $\therefore AB \rightarrow C$ 非多余的, 不能去掉.

②: 设 $CG \rightarrow B$ 为多余的, 则去掉 $CG \rightarrow B$ 得 $F_2 = \{AB \rightarrow C, C \rightarrow A, CG \rightarrow D, ACD \rightarrow B\}$

求 $(CG)_{F_2}^+$. 设 $X^{(0)} = CG$, i: 计算 $X^{(1)}$: 逐一扫描 F_2 中的各个函数依赖, 找到左部为 C, G 或 CG 的函数依赖, 得 $C \rightarrow A$, $\therefore X^{(1)} = CGA$.

ii: 计算 $X^{(2)}$: \because 有 $CG \rightarrow D$, $\therefore X^{(2)} = ACDG$;

iii: 计算 $X^{(3)}$: \because 有 $ACD \rightarrow B$, $\therefore X^{(3)} = ABCDG$.

iv: 计算 $X^{(4)}$: \because 有 $X^{(4)} = X^{(3)}$, $\therefore (CG)_{F_2}^+ = ABCDG$.

又: $CG \rightarrow B$ 的右部 $B \subset ABCDG = (CG)_{F_2}^+$, $\therefore CG \rightarrow B$ 为多余的, 去掉.

得 $F_2 = \{AB \rightarrow C, C \rightarrow A, CG \rightarrow D, ACD \rightarrow B\}$.

③: 设 $CG \rightarrow D$ 为多余的, 则去掉得 $F_3 = \{AB \rightarrow C, C \rightarrow A, ACD \rightarrow B\}$.

求 $(CG)_{F_3}^+$. 设 $X^{(0)} = CG$; i: 计算 $X^{(1)}$: \because 有 $C \rightarrow A$, $\therefore X^{(1)} = ACG$;

ii: 计算 $X^{(2)}$: 有得 $X^{(2)} = X^{(1)}$, $\therefore (CG)_{F_3}^+ = ACG$.

$\because CG \rightarrow D$ 的右部 $D \notin ACG = (CG)_{F_3}^+$,

$\therefore CG \rightarrow D$ 不是多余的函数依赖, 不能去掉.

④: 设 $ACD \rightarrow B$ 为多余的, 则去掉得 $F_4 = \{AB \rightarrow C, C \rightarrow A, CG \rightarrow D\}$.

求 $(ACD)_{F_4}^+$: $X^{(0)} = ACD$, $X^{(1)}$: \because 有 $C \rightarrow A$, $\therefore X^{(1)} = ACD$, $X^{(2)} = X^{(1)}$

又: $ACD \rightarrow B$ 的右部 B 不在 $(ACD)_{F_4}^+ = ACD$ 中, $\therefore ACD \rightarrow B$ 不是多余的, 不能去掉.

(3): 去掉 $F_4 = \{AB \rightarrow C, C \rightarrow A, CG \rightarrow D, ACD \rightarrow B\}$ 中各依赖左部的多余属性:

\because 有 $C \rightarrow A$, 因此, 在 $ACD \rightarrow B$ 中, 属性 A, C 在同-依赖左、右部出现,

$\therefore ACD \rightarrow B$ 的属性 A 是多余的. 去掉.

得最小函数依赖集: $F_{min} = \{AB \rightarrow C, C \rightarrow A, CG \rightarrow D, CD \rightarrow B\}$.

数据库系统概述----最小函数依赖集求解过程 - Little_Lu - 博客园 (cnblogs.com)

数据库原理: 求最小依赖集和候选键 - 乌漆WhiteMoon - 博客园 (cnblogs.com)

模式分解 !

模式分解的三个定义


- 分解具有无损连接性
- 分解要保持函数依赖
- 分解既要保持函数依赖又要具有无损链接行

判断分解是否具有无损连接性（表格法） ?


数据库手把手解题——1.判断无损连接_数据库无损连接的判断-CSDN博客 (怎么建表)

判断是否为无损连接分解_数据库u=(a,b,c,d,e,f),f={a→bc}-CSDN博客

- ① 建立一张n列k行的表，每一列对应一个属性，每一行对应分解中的一个关系模式。若属性 $A_j \in U_i$ ，则在j列i行填上 a_j ，否则填上 b_{ij} ；
- ② 对于每一个FDi做如下操作：找到Xi所对应的列中具有相同符号的那些行。考察这些行中li列的元素，若其中有 a_j ，则全部改为 a_j ，否则全部改为 b_{mli} ，m是这些行的行号最小值。



Lossless Decomposition and Functional Dependencies (additional)





[例 6.15] 已知 $R < U, F >$, $U = \{A, B, C, D, E\}$, $F = \{AB \rightarrow C, C \rightarrow D, D \rightarrow E\}$, R 的一个分解为 $R_1(A, B, C)$, $R_2(C, D)$, $R_3(D, E)$ 。

(1) 首先构造初始表

	A	B	C	D	E
ABC	a_1	a_2	a_3	b_{14}	b_{15}
CD	b_{21}	b_{22}	a_3	a_4	b_{25}
DE	b_{31}	b_{32}	b_{33}	a_4	a_5

(2) 对 $AB \rightarrow C$ ，因各元组的第一、二列没有相同的分量，所以表不改变。由 $C \rightarrow D$ 可以把 b_{14} 改为 a_4 ，再由 $D \rightarrow E$ 可使 b_{15} 、 b_{25} 全改为 a_5 。最后结果为图 6.9 (b)。表中第一行成为 a_1 、 a_2 、 a_3 、 a_4 、 a_5 ，所以此分解具有无损连接性。

A	B	C	D	E
a_1	a_2	a_3	a_4	a_5
b_{21}	b_{22}	a_3	a_4	a_5
b_{31}	b_{32}	b_{33}	a_4	a_5



典型题型

- 画函数依赖图

[例 6.1] 建立一个描述学校教务的数据库，该数据库涉及的对象包括学生的学号 (Sno)、所在系 (Sdept)、系主任姓名 (Mname)、课程号 (Cno) 和成绩 (Grade)。假设用一个单一的关系模式 Student 来表示，则该关系模式的属性集合为

$$U = \{Sno, Sdept, Mname, Cno, Grade\}$$

现实世界的已知事实 (语义) 告诉我们：

- ① 一个系有若干学生，但一个学生只属于一个系。
- ② 一个系只有一名 (正职) 负责人。
- ③ 一个学生可以选修多门课程，每门课程有若干学生选修。
- ④ 每个学生学习每一门课程有一个成绩。

于是得到属性组 U 上的一组函数依赖 F (如图 6.1 所示)。

$$F = \{Sno \rightarrow Sdept, Sdept \rightarrow Mname, (Sno, Cno) \rightarrow Grade\}$$

如果只考虑函数依赖这种数据依赖，可以得到一个描述学生的关系模式 $Student \langle U, F \rangle$ 。表 6.1 是某一时刻关系模式 Student 的一个实例，即数据表。

表 6.1 Student 表

Sno	Sdept	Mname	Cno	Grade
S1	计算机系	张明	C1	95
S2	计算机系	张明	C1	90
S3	计算机系	张明	C1	88
S4	计算机系	张明	C1	70
S5	计算机系	张明	C1	78
⋮	⋮	⋮	⋮	⋮

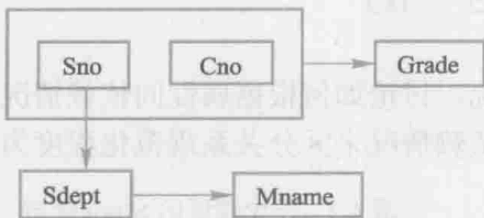


图 6.1 Student 上的一组函数依赖

• P182-P184 定义6.6, 6.7 (2NF, 3NF)

• 求闭包

[例 6.11] 已知关系模式 $R \langle U, F \rangle$ ，其中

$$U = \{A, B, C, D, E\}, F = \{AB \rightarrow C, B \rightarrow D, C \rightarrow E, EC \rightarrow B, AC \rightarrow B\}.$$

求 $(AB)^+_F$ 。

解 由算法 6.1，设 $X^{(0)} = AB$ 。

计算 $X^{(1)}$ ：逐一扫描 F 集合中各个函数依赖，找左部为 A 、 B 或 AB 的函数依赖。得到两个： $AB \rightarrow C$ ， $B \rightarrow D$ 。于是 $X^{(1)} = AB \cup CD = ABCD$ 。

因为 $X^{(0)} \neq X^{(1)}$ ，所以再找出左部为 $ABCD$ 子集的那些函数依赖，又得到 $C \rightarrow E$ ， $AC \rightarrow B$ ，于是 $X^{(2)} = X^{(1)} \cup BE = ABCDE$ 。

因为 $X^{(2)}$ 已等于全部属性集合，所以 $(AB)^+_F = ABCDE$ 。

对于算法 6.1，令 $a_i = |X^{(i)}|$ ， $\{a_i\}$ 形成一个步长大于 1 的严格递增的序列，序列的上界是 $|U|$ ，因此该算法最多 $|U| - |X|$ 次循环就会终止。

- 求最小函数依赖集

[例 6.13] $F=\{A\rightarrow B, B\rightarrow A, B\rightarrow C, A\rightarrow C, C\rightarrow A\}$

$$F_{m1}=\{A\rightarrow B, B\rightarrow C, C\rightarrow A\}$$

$$F_{m2}=\{A\rightarrow B, B\rightarrow A, A\rightarrow C, C\rightarrow A\}$$

这里给出了 F 的两个最小依赖集 F_{m1} 、 F_{m2} 。

若改造后的 F 与原来的 F 相同，说明 F 本身就是一个最小依赖集，因此定理 6.3 的证明给出的极小化过程也可以看成是检验 F 是否为极小依赖集的一个算法。