CH4 Relational Database Design Theory关系数据库设计理论

一、Functional Dependencies

1. 定义

$A_1A_2A_3\cdots A_n \rightarrow B_1B_2B_3\cdots B_m$

$\rightarrow$为函数映射，即$a = b \Rightarrow f(a) = f(b)$

R satisfies the FD F

2. 键

能够函数映射关系中所有属性的**最小**原象集。

3. 主键

关系可能不止一个键，把所有键中的一个称为主键。

4. 超键

包含键的集合。

5. 函数依赖集

set of FD' S:

$\{a \rightarrow b, a \rightarrow d, b \rightarrow c, d \rightarrow b\}$

6. 等价Equivalent

两个函数依赖集S和T等价，当且仅当对任意的关系，如果S在该关系上成立，T也在该关系上成立，反之也成立。

7. 推断Follows

S follows T 即 $T \rightarrow S$

两个函数依赖集S和T若具有推断关系$S \Rightarrow T$

则对任意关系，若在该关系S成立，则在该关系上T也成立，反之未必成立。

8. 平凡函数依赖

$As \rightarrow Bs$

且$Bs \subseteq As$

二. 定理

1. 分解规则/组合规则

$A_1A_2\cdots A_n\rightarrow B_1B_2\cdots B_n$

$\iff A_1A_2\cdots A_n\rightarrow B_1\wedge A_1A_2\cdots A_n\rightarrow B_2\cdots$

***仅能分解/组合→右侧的属性

2. 传递规则

$As\rightarrow Bs, Bs\rightarrow Cs\Rightarrow As\rightarrow Cs$

***使用闭包证明：

$As\rightarrow Bs\Rightarrow Bs\subseteq As^+$

$Bs\rightarrow Cs\Rightarrow Cs\subseteq As^+$

则$As\rightarrow Cs$成立

3. 平凡依赖性

$As\rightarrow Bs\iff As\rightarrow Cs$

$Cs=Bs-As$

即，对右侧属性去掉原象部分函数依赖依然成立

4. 闭包Closure

$As=A_1A_2\cdots A_n$为属性集合

S为函数依赖

As在S下的闭包As^+

且$As^+=\{B|S\Rightarrow As\rightarrow B\}$

显然，$As\subseteq As^+$

```python
def Closure(As, S):
    res = As
    for s in S:
        S.insert(split(s))  # 将As -> Bs的右侧拆分为一个元素
    while 1:
        for bs in res:
            find(bs, S, c)  # 找到Bs -> c的函数依赖
            res.insert(c)
        if nothing found:
            break
    return res
```

*** $As\rightarrow Bs$满足函数依赖S当且仅当$Bs\subseteq As^+$

5. 闭包和键

As^+为关系的全部属性当且仅当As为超键

***判断某一Cs是否为键：

> 1. 判断Cs^+是否为关系的全部属性
> 2. 判断Cs的某个真子集cs的闭包是否为关系的全部属性
> 3. 不存在上述真子集则Cs为键
> 4. 存在上述真子集则Cs为超键

6. 最小基本集Minimal Basis

给定函数依赖S，任意与S等价的函数依赖都称作S的基本集。

最小基本集：

> 每条函数依赖右边只有一个属性
>
> 删除任意一条函数依赖，得到的集合不是基本集
>
> 删除任意函数依赖左侧中任意的属性，得到的集合不是基本集

7. Armstrong's Axioms公理

7.1 Reflexivity自反律

与平凡函数依赖一致

7.2 Augmentation增广律

$$As \to Bs \Rightarrow AsCs \to BsCs$$

7.2 Transitivity传递律

$$As \to Bs, Bs \to Cs \Rightarrow As \to Cs$$

8. 函数依赖的投影

关系R，函数依赖S，$\pi_L(R)$的函数依赖称为函数依赖S的投影

***同样可以用下列算法求解最小基本集

```
def FD_Projection(R, S, R1):
    res = None
    for r in subset(R1):
        r_plus = closure(r, S)
        for nontrivel_r(x, a) in r_plus:     # 对所有非平凡依赖x->a, x属于r, a属于R1 交 r+
            res.insert(x, a)
    # 现在T为R1中的一个基本集
    for fd in res:
        if(fd could be followed by other fds in res):
            res.delete(fd)
    for dupfd in res:   # 对所有左侧大于1个属性的函数依赖
        for x in left_spilit(dupfd) # 将左侧拆分得到的函数依赖
            if(x coule be folowed by other fds in res):
                res.replace(dupfd, x)
    return res
```

三、函数依赖范式

redundancy 冗余

anomaly 异常

1. Anomalies异常

> 1. Redundancy冗余
> 2. Update Anomalies更新异常
> 3. Deletion Anomalies删除异常

2. Decomposing分解

给定关系$R(As)$，若$S(Bs)$和$T(Cs)$满足下列三个条件，则称$S(Bs)$和$T(Cs)$是$R(As)$一个分解

> 1. $As = Bs \cup Cs$
> 2. $S = \pi_{Bs}(R)$
> 3. $T = \pi_{Cs}(R)$

3. BCNF

一个关系R满足BC范式当且仅当对于任意非平凡函数依赖$As \rightarrow Bs$，As一定为超键

***任意具有两个属性的关系满足BC范式

$-R(A, B)$

四种情况讨论即可完成证明

4. 分解为BC范式

```
def BCDecomposition(R0, S0):
    R, S = R0, S0
    if(isBC(R)):
        return R
    else:
        vio_fd(X, Y) = violations(R, S)      # X->Y为任意冲突的函数依赖
        X_plus = closure(X, S)               # 计算X的闭包
        R1 = X_plus                          # X+
        R2 = X union (R - X_plus)            # 在R中却不在X+中的
        S1 = FD_Projection(R0, S, R1)        # 函数依赖在R1上的投影
        S2 = FD_Projection(R0, S, R2)        # 函数依赖在R2上的投影
        return BCDecomposition(R1, S1) union BCDecomposition(R2, S2)
```

5. 分解的目的

> 1. 消除冗余
> 2. 提高信息可恢复性
> 3. 保留函数依赖