

# Database - P12

外连接

(outer join), 作用于关系R和S的二元运算

与内连接不同，外连接额外包含由一个关系中的未匹配元组

扩展“填充”值到另一个关系的每个属性上而获得的元组

additionally tuple formed by extending an unmatched tuple in one operand

by "fill" value for each attribute of the other operand

外连接定义空值(null value) w 的存在，

在实际应用中空值对应于SQL语言中的NULL

in practice correspond to the NULL in SQL

为了使作用在结果关系上的后续选择运算具有意义

需要对空值 null 赋予语义上的意义

make subsequent selection operation on the resulting table meaningful

a semantic meaning needs to be assigned to null

于是命题逻辑扩展为三值逻辑，用于选择运算

the propositional logic used by selection extend to three-valued logic

注意 left outer join, right outer join, full outer join 中的 outer 有时会省略

左外连接(left outer join), 运算符为  $\Delta L$ , 如  $R \Delta L S$

在其结果中，除了在公共属性上相等的元组的组合，还包括 R 中未匹配到 S 的元组

关系 S 中在公共属性上没有公共值的元组置为空值 null (w).

通过关系代数原始运算模拟左外连接

对于  $r_1, r_2, \dots, r_m$  为关系 R 的特有属性,  $(w, w, \dots, w)$  为定义在 S 特有属性上的空值元组

则左外连接可以基于自然连接定义

$$R \Delta L S = R \bowtie S \cup ((R - \pi_{r_1, r_2, \dots, r_m}(R \bowtie S)) \times \{(w, w, \dots, w)\})$$

右外连接(right outer join), 运算符为  $\Delta R$ , 如  $R \Delta R S$

在其结果中，除了在公共属性上相等的元组的组合，还包括 S 中未匹配到 R 的元组

关系 R 中在公共属性上没有公值的元组置为空值 null (w)

右外连接可以基于自然连接定义，通过关系代数原始运算模拟

对于  $s_1, s_2, \dots, s_n$  为关系 S 的特有属性,  $(w, \dots, w)$  为定义在 R 特有属性上的空值元组

$$R \Delta R S = R \bowtie S \cup (\{(w, w, \dots, w)\} \times (S - \pi_{s_1, s_2, \dots, s_n}(R \bowtie S)))$$

全外连接(full outer join), 运算符为  $\Delta F$ , 如  $R \Delta F S$

其结果为左外连接与右外连接的并集，即  $R \Delta F S = R \Delta L S \cup R \Delta R S$

注意在 SQL 语言中，提供了关键词左外连接(left join)和右外连接(right join)

但全外连接需要通过 left join ... union ... right join 实现

# Database - P13

三值逻辑 (three-valued logic, 3VL), 又称三元 (ternary) 逻辑或三价 (trivalent) 逻辑

相比二元逻辑, 包含真 (TRUE), 假 (FALSE), 不确定的第三值 (indeterminate third value)

常用不同的三进制系统 (ternary numeral system) 表示

如三进制数 (ternary numeral system), 每一位数值为 0, 1, 2

平衡三进制 (balanced ternary), 每一位为 -1, 0, +1 或简化为 -1, 0, +

1 表示 TRUE, 2 表示 FALSE, 0 表示未知 (Unknown), 无关 (irrelevant), 皆有 (both)

1 表示 TRUE, 0 表示 FALSE, 非整型数符号表示 "maybe", 如 ?, #, 1/2, xy

在布尔逻辑中, 有  $2^2 = 4$  种不同的一元运算符, 有  $2^{2^2} = 16$  种不同的二元运算符

但在三值逻辑中, 有  $3^3 = 27$  种不同的一元运算符, 有  $3^{3^2} = 19683$  种不同的二元运算符

所以布尔逻辑中的 ~~一元~~ 运算 ( $\neg, \wedge, \vee, \top, \downarrow, \oplus, \leftrightarrow, \rightarrow$ ) 无法直接应用于三值逻辑

与布尔逻辑类似, 可以用真值表来表示运算符的运算结果

与布尔逻辑一样, 在真值表中用 T 表示 TRUE, 用 F 表示 FALSE

但是关于 U 表示的 UNKNOWN 有不同的解释

Kleene 逻辑 (Kleene logic) 中, unknown 表示既非真也非假 (neither true nor false)

于是 Kleene 逻辑中, 真值 (~~truth value~~) 仅指派 (designated) 给 TRUE

Priest 逻辑 (Priest logic) 中, unknown 表示既真也假 (both true and false)

于是在 Priest 逻辑中, 真值被指派于 TRUE 和 UNKNOWN

在 Kleene 逻辑中, 关于 unknown 在某一时刻隐含着 true 或 false 的信息是不可知的

the knowledge of whether any particular unknown state

secretly represents true or false at any moment in time is not available

真值表

	NOT A	$\neg A$	AND. A $\wedge$ B	$A \wedge B$	OR AVB	$A \vee B$
T	F	T	F	F	T	T
U	U	U	F	F	U	U
F	T	F	F	F	T	F

也可以在平衡三进制上定义为 neg(A), min(A, B), max(A, B) 函数

$A \text{ neg}(A) \quad ; \min(A, B) \quad ; \max(A, B) \quad ; \quad +1 \quad 0 \quad -1$

+1    -1    ;    1    0    -1    ;    1    0    +1    +1    +1

0    0    0    ;    0    1    0    ;    0    -1    0    ;    0    0    +1

-1    +1    -1    ;    -1    -1    -1    ;    -1    -1    +1    0    -1

并且可以进一步扩展为  $A_1 \wedge A_2 \wedge \dots \wedge A_n = \min(A_1, A_2, \dots, A_n)$

$A_1 \vee A_2 \vee \dots \vee A_n = \max(A_1, A_2, \dots, A_n)$

# Database - P14

三值蕴含 在 Kleene logic 中，蕴含 (implication) 定义为  $A \rightarrow B \stackrel{\text{def}}{=} \text{OR}(\text{NOT}(A), B)$

即有  $A \rightarrow B \equiv T \vee U \vee F$  或  $A \rightarrow B \equiv +1 \vee 0 \vee -1$

$T \rightarrow T \equiv T \vee U \vee F \equiv +1 \vee +1 \vee 0 \equiv +1 \equiv 1$

$U \rightarrow T \equiv T \vee U \vee U \equiv 0 \vee +1 \vee 0 \equiv 0 \equiv 0$

$+1 \rightarrow T \equiv T \vee T \equiv -1 \vee +1 \equiv +1 \equiv 1$

$F \rightarrow T \equiv T \vee F \equiv -1 \vee -1 \equiv -1 \equiv -1$

(注)  $\text{IMP}_k(A, B) \equiv \text{OR}(\neg A, B) \equiv \max(-A, B)$

注意：在 Kleene logic 中不存在永真式 (tautology)，或称 valid formula

由于当对于一个良定义的复合命题的所有原子命题被指派 UNKNOWN 时

复合命题本身的真值必定为 UNKNOWN

而在 Kleene logic 中只有 TRUE 被指派了 truth value

all atomic components of well-formed formula are assigned UNKNOWN  
formula itself must also have value UNKNOWN

但是注意，对于 Kleene logic 而言，缺少永真式并不表示缺少有效论证或推理规则 rules

lack of valid formula doesn't mean lack of valid argument and/or inference

在 Kleene logic 中的语义有效的论证 (semantically valid argument)

只要其所有前提是真，则其结论必定为真

whenever (for any interpretation/model) all of premises are TRUE

the conclusion must also be TRUE

另外，对于 Logic of Paradox (LP)，存在永真式 (analogue of BOTH)

与 Kleene Logic 相比，拥有相同的真值表，但指派 truth value 为 TRUE 和 UNKNOWN

但是拥有更少的推理规则 (fewer inference rule)

SQL 应用 在 SQL 中， $T$  为处理内容为 NULL 的属性 (comparison with NULL field content)

注意 NULL 值的意义，在数据库中缺失数据，但本身真实存在 but not recorded  
represent missing data in database, assumption that actual value exists

SQL 使用 Kleene K3 logic 中的通用部分 (common fragment)

即仅有 AND, OR, NOT 的真值表

在 SQL 中，中间值被认为应解释为 UNKNOWN

intermediate value intended to be interpreted as UNKNOWN

对于 NULL 值与任何其他值比较，都应有结果为 UNKNOWN

但 SQL 中实际上将 NULL 认为是一种实际存在的值

即在 UNION 或 INTERSECT 中，NULL 被认为是 equal with each other

# Database - P15

Lukasiewicz 逻辑, Lukasiewicz K3 logic 使用与 Kleene K3 logic 相同的 AND, OR, NOT 真值表

但是对蕴含的定义有所不同

表现在 K3 logic 定义 "UNKNOWN implies UNKNOWN" 命题为 TRUE

即有  $\text{IMP}_K(A, B) \equiv \min(C_1, 1 - A + B)$

$$A \rightarrow B \begin{array}{c|cc} T & 1 & 0 \\ F & 0 & 1 \end{array} \quad A \rightarrow B \begin{array}{c|cc} T & 1 & 0 \\ F & 0 & 1 \end{array}$$

$$T \quad T \quad U \quad F \quad 1 - 1 + 1 = 0 \quad 1 - 0 + 1 = 1$$

$$U \quad T \quad T \quad U \quad 1 - 0 + 1 = 0 \quad 1 - 0 + 1 = 0$$

$$F \quad 0 \quad T \quad T \quad 1 - 1 + 1 = 1 \quad 1 - 1 + 1 = 1$$

基于  $\text{IMP}_K$  的定义, 可以构造逻辑等价式

$$A \vee B \equiv (A \rightarrow B) \rightarrow B$$

注意在 K3 logic 中,  ~~$T \vee U \equiv T$~~

$$\text{但 } (T \rightarrow U) \rightarrow U \equiv U \rightarrow U \equiv U \neq T \vee U$$

但在 K3 logic 中,  $(T \rightarrow U) \rightarrow U \equiv U \rightarrow U \equiv T \equiv T \vee U$

$$A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A)$$

注意在 K3 logic 中,  $U \leftrightarrow U$  实际上应等于 T

$$\text{但 } (U \rightarrow U) \wedge (U \rightarrow U) \equiv U \wedge U \equiv U \neq T$$

但在 K3 logic 中,  $(U \rightarrow U) \wedge (U \rightarrow U) \equiv T \wedge T \equiv T \equiv U \leftrightarrow U$

基于  $\text{IMP}_K$  的定义, 还可以构造一元运算符

$$MA \equiv \neg A \rightarrow A, \text{ 称为 "it is not false that", 即 "A 非假"}$$

$$LA \equiv \neg \neg A, \text{ 称为 "it is true / necessary that", 即 "A 为真"}$$

$$IA \equiv MA \wedge LA, \text{ 称为 "it is unknown / contingent that", 即 "A 未定"}$$

即有  $A \quad MA \quad LA \quad IA \quad A \quad MA \quad LA \quad IA$

$$T \quad T \quad T \quad F \quad 1 \quad 1 \quad 1 \quad -1$$

$$U \quad T \quad F \quad T \quad 0 \quad +1 \quad -1 \quad +1$$

$$F \quad F \quad F \quad F \quad -1 \quad -1 \quad -1 \quad -1$$

均不是永真式

在 K3 logic 中, law of excluded middle ( $A \vee \neg A$ ) 和 law of non-contradiction ( $\neg(\neg A \wedge \neg A)$ )

于是引入: law of excluded fourth ( $A \vee IA \vee \neg A$ )

extended contradiction principle ( $\neg(\neg A \wedge \neg IA \wedge A)$ )

在 SQL 语言中, IS UNKNOWN 对于 K3 logic 中的运算符 I, 使得三值逻辑功能上完整

can express (in combination) any conceivable three-valued logical function

# Database - P16

第9章 - 第二部分

聚集函数 (Aggregate function), 或 aggregation function)

指将若干行的数据组合起来以形成一个单一汇总值的函数

values of multiple rows are grouped together to form a single summary value

常用的聚集函数有

		返回值 (arithmetic mean)	SQL 实现
Average	平均值	数据组的算术平均值	AVG (expression), 其中 expression 为字段
Count	计数	数据组中的数据个数	COUNT (expression), expression 为字段或 *
Maximum	最大值	数据组中的最大值	MAX (expression), expression 为字段
Minimum	最小值	数据组中的最小值	MIN (expression), expression 为字段
Median	中位数	数据组中的中位数	—
Mode	众数	数据组中的众数	—
Range	范围	数据组中最大值与最小值之差	—
Sum	求和	数据组中数据的和	SUM (expression), expression 为字段
Nanmean	非空平均值	忽略非数字值 (Not a Number, NaN), 如 nil 或 NULL 后的平均值	—
Stddev	标准差	数据组的标准差	—

注意：在 MySQL 中，AVG() 函数是自动忽略 NULL 值的。

而在其他数据库中，可能存在将 NULL 视为 0 或其他特殊值的情形

如在 MySQL 中，age: 11 22 NULL 33 44

select AVG(age) → 27.50

而 age: 11 22 0 33 44

select AVG(age) → 22.00

在关系代数中，对于关系 R,  $r_1, r_2, \dots, r_n$  为其特有属性

则聚集函数可写作形如  $f_1, f_2, \dots, f_m g(r'_1), f(r'_2), \dots, f(r'_n)$  ( $R$ ) 的表达式

其中  $\{r'_1, r'_2, \dots, r'_n\}$  为  $\{r_1, r_2, \dots, r_n\}$  的子集

$g$  表示 grouping attribute, 即对关系 R 按 ~~group by clause~~ 分组

等价于 SQL 中的 ~~group by clause~~

也可以写作  $exp_1, exp_2, \dots, exp_m | func_1, func_2, \dots, func_n (R)$

局限性

存在关系上简单的运算无法用关系代数实现，如关系 R 的传递闭包 (transitive closure)

即对于关系代数表达式  $E(R)$ , 存有结果  $R^+$ , 使得  $R \subseteq R^+$  且  $\forall x \forall y \forall z [(x,y) \in R \wedge (y,z) \in R^+ \rightarrow (x,z) \in R^+]$

可以证明对于变量关系 R, 总能找到 R 的实例 r, 使得  $E(r) \neq r^+$

can always find an instance r of R, and a corresponding domain d

such that  $E(r) \neq r^+$

# Database - P17

查询优化的代数性质 (algebraic property for query optimization)

查询可以表示为树 (query can be represented as tree)

内部节点表示运算符 (internal nodes are operators)

叶结点表示关系 (leaves are relations)

子树表示子表达式 (subtrees are subexpressions)

查询优化的过程是把表达式树转换为等价的表达式树

transform expression tree into equivalent expression tree

使得树中子表达式生成的关系的平均大小，小于优化前的

average size of relations yielded by subexpression in the tree

smaller than before optimization

其次要目标是在单一查询中，或在同时求值的多个查询中，形成公共子表达式

to form common subexpressions within a single query

or in all of more than one query being evaluated at the same time

即公共子表达式仅需计算一次，而结果可用于所有包含子表达式的查询

enough to compute common subexpression once

result can be used in all queries containing the subexpression

选择，在查询优化中，选择运算可以有效地减少其操作数中的行数

于是在优化过程中，应尽量将选择运算向叶结点方向移动

move the selections in an expression tree towards the leaves

从而将子表达式产生的内部关系 (internal relation) 缩小

幂等律 (idempotent) 指多次运用同一选择运算并不产生额外效果

multiple applications of same ~~selection~~ have no additional effect beyond first one

即有  $\sigma_A \sigma_A(R) = \sigma_A(R)$ ，或者有  $(\sigma_A)^n(R) = \sigma_A(R)$ ，其中  $n \in \mathbb{Z}^+$

交换律 (commutative)，指应用选择的顺序不影响最终结果

即有  $\sigma_A \sigma_B(R) = \sigma_B \sigma_A(R)$

分离复杂条件 (break with complex conditions)

指可以将复杂条件拆分为简单条件的合并来减少需要求值的选择运算

merge selections so that fewer selections need to be evaluated

或可以分解为更简单的选择运算使得可以单独地移动和优化

separately

split selections so that component selections may be moved or optimized

# Database - P18

## 查询优化

简单条件的合取 (conjunction) 等价于顺序地进行一系列简单选择

即有  $\sigma_{A \wedge B}(R) = \sigma_A(\sigma_B(R)) = \sigma_B(\sigma_A(R))$

简单条件的析取 (disjunction) 等价于简单选择结果的并集

即有  $\sigma_{A \vee B}(R) = \sigma_A(R) \cup \sigma_B(R)$

对于选择与叉积 (cross product operator)

由于叉积对于输入为 N 行与 M 行的关系，输出为 NM 行

所以是耗费最大 (costliest) 的运算，减少其操作数的大小是重要的

当叉积运算后跟随一个选择运算，即形如  $\sigma_A(R \times S)$  时，

根据在选择运算中分离复杂条件 (split rule)

令  $A = B \wedge C \wedge D$ , B, C, D 为不同的简单条件

其中 B 包含 ~~只在 R 中的属性~~ 只在 R 中的属性,  $r_1, r_2, \dots, r_k$

C 包含 S 的特有属性,  $s_1, s_2, \dots, s_l$

D 包含 R 和 S 的公共属性,  $c_1, c_2, \dots, c_m$

则可知 B, C, D 是不相交的，且 B, C, D 均有可能为空

于是有  $\sigma_A(R \times S) = \sigma_B \sigma_C \sigma_D(R \times S)$

$= \sigma_D(\sigma_B \sigma_C(R \times S))$

$= \sigma_D(\sigma_B \sigma_C(R) \times \sigma_B \sigma_C(S))$

$= \sigma_D(\sigma_B(R) \times \sigma_C(S))$

对于选择与其他集合运算，如差集 (set difference), 交集 (intersection), 并集 (union)

选择运算对于这些集合运算是分配性 (distributive) 的

特别地有，对于差集和交集运算，有时可以只对一个运算数执行选择

由于当其中一个运算数较小时，对其进行选择运算的耗费大于直接使用的好处

即有  $\sigma_A(R \setminus S) = \sigma_A(R) \setminus \sigma_A(S) = \sigma_A(R) \setminus S$

$\sigma_A(R \cap S) = \sigma_A(R) \cap \sigma_A(S) = \sigma_A(R) \cap S$

这里 S 即为含有较少元组的关系

另外  $\sigma_A(R \cup S) = \sigma_A(R) \cup \sigma_A(S)$

(commute)

对于选择与投影运算，可交换当且仅当选择的条件是投影范围的子集

由于投影会去除重复的元组而使得选择运算只需检查更少的元组

即有  $\pi_{a_1, \dots, a_n}(\sigma_A(R)) = \sigma_A(\pi_{a_1, \dots, a_n}(R))$ , 其中  $A \subseteq \{a_1, a_2, \dots, a_n\}$

注意，当操作数为叉积或连接时，选择在投影之前有时是有意义的

# Database - P19

## 查询优化

在查询优化中，投影与选择相似，也是幂等性 (Idempotent) 的  
一系列有效的投影运算等价于最外侧的投影运算

a series of valid projections is equivalent to the outermost projection

即有  $\pi_{a_1, \dots, a_n}(\pi_{b_1, \dots, b_m}(R)) = \pi_{a_1, \dots, a_n}(R)$ , 其中  $\{a_1, \dots, a_n\} \subseteq \{b_1, \dots, b_m\}$

注意有效的投影运算  $\pi_{a_1, \dots, a_n}(R)$  中, R 的属性集合为  $\{r_1, r_2, \dots, r_m\}$

则应有  $\{a_1, a_2, \dots, a_n\} \subseteq \{r_1, \dots, r_m\}$

于是如果一系列投影运算是有效的,

如  $\pi_A(\pi_B(\pi_C(R)))$ , 则应有  $A \subseteq B \subseteq C \subseteq \{r_1, \dots, r_m\}$

投影与选择的不同在于, 其分配律 (distributive) 仅作用于并集, 而交集与差集不可用

即有  $\pi_{a_1, \dots, a_n}(R \cup P) = \pi_{a_1, \dots, a_n}(R) \cup \pi_{a_1, \dots, a_n}(P)$ , 其中  $\{a_1, \dots, a_n\} \subseteq \{r_1, \dots, r_m\}$

但是对于交集和差集则不满足分配律

如 给定关系  $R = \{(A=a, B=b)\}$ ,  $P = \{(A=a, B=b')\}$ , 其中  $b \neq b'$

则  $\pi_A(R \cap P) = \pi_A(\{(A=a, B=b)\} \cap \{(A=a, B=b')\})$

$= \pi_A(\emptyset) = \emptyset$

而  $\pi_A(R) \cap \pi_A(P) = \pi_A(\{(A=a, B=b)\}) \cap \pi_A(\{(A=a, B=b')\})$

$= \{(A=a)\} \cap \{(A=a)\} = \{(A=a)\}$

于是交集的投影不等于投影的交集

另外  $\pi_A(R \setminus P) = \pi_A(\{(A=a, B=b)\} \setminus \{(A=a, B=b')\})$

$= \pi_A(\{(A=a, B=b)\}) = \{(A=a)\}$

而  $\pi_A(R) \setminus \pi_A(P) = \pi_A(\{(A=a, B=b)\}) \setminus \pi_A(\{(A=a, B=b')\})$

$= \{(A=a)\} \setminus \{(A=a)\} = \emptyset$

于是差集的投影不等于投影的差集

对于重命名运算, 对于同一变量的连续 (successive) 重命名可以压缩 (collapse) 为一次重命名

即有  $\text{Par}_b(\text{Par}_c(R)) = \text{Par}_c(R)$

而对于没有公共变量的重命名运算, 可以任意排序, 使得同一变量的重命名连续并可压缩

即有  $\text{Par}_b(\text{Par}_{xy}(\text{Par}_c(R))) = \text{Par}_b(\text{Par}_c(\text{Par}_{xy}(R))) = \text{Par}_c(\text{Par}_{xy}(R))$

对于重命名运算与集合运算, 与选择相似, 对并集、交集、差集都是服从分配律的

即有  $\text{Par}_b(R \cup P) = \text{Par}_b(R) \cup \text{Par}_b(P)$

$\text{Par}_b(R \cap P) = \text{Par}_b(R) \cap \text{Par}_b(P)$

$\text{Par}_b(R \setminus P) = \text{Par}_b(R) \setminus \text{Par}_b(P)$

# Database - P<sub>20</sub>

查询优化 对于笛卡尔积，对于并集运算是服从分配律的  
即有  $(A \times B) \cup (A \times C) = A \times (B \cup C)$

函数依赖性 (functional dependency)，指在关系型数据库中关系的两组属性间的约束 (constraint)

对于给定的关系 R，其属性集合为  $\{r_1, r_2, \dots, r_n\}$

则对于属性集  $X, Y \subseteq \{r_1, r_2, \dots, r_n\}$

如果关系中的两个元组，在属性集合 X 上有相同的值，则在属性 Y 上必有相同的值

称属性集合 X 函数确定 (functionally determine) 属性集合 Y

记为  $FD: X \rightarrow Y$ ，并称关系 R 满足 (satisfy) 函数依赖性  $FD: X \rightarrow Y$

称 X 为决定因素 (determinant set)，Y 为依赖因素 (dependent set)

如果  $Y \subseteq X$ ，则称  $FD: X \rightarrow Y$  为平凡 (trivial) 函数依赖

由于对于任意关系 R 的任意属性集合 X，平凡函数均成立

所以在不特别声明的情况下，均讨论非平凡函数依赖

如果有  $X \rightarrow Y$  且  $Y \rightarrow X$ ，则记为  $X \leftrightarrow Y$

如果有  $Y$  不函数依赖于  $X$ ，则记为  $X \not\rightarrow Y$

完全函数依赖 (full functional dependency)，指对于关系 R 中的属性集合 X 和 Y

如果有  $X \rightarrow Y$ ，且  $\forall X' \subset X \quad X' \not\rightarrow Y$ ，即  $Y$  不函数依赖于任何 X 的真子集  $X'$   
称 Y 对 X 完全函数依赖，记作  $X \xrightarrow{F} Y$

也可以说从 X 中移去任何属性而形成的集合  $X'$ ，不再满足  $X' \rightarrow Y$

if removal of any attribute  $r_i$  from X

means that the dependency doesn't hold any more

部分函数依赖 (partial functional dependency)，指对于关系 R 中的属性集合 X 和 Y

如果有  $X \rightarrow Y$ ，且  $\exists (X \xrightarrow{F} Y)$ ，即  $\exists X' \subset X \quad X' \rightarrow Y$

称 Y 对 X 部分函数依赖，记作  $X \xrightarrow{P} Y$

也可以说 Y 对 X 不完全函数依赖

传递函数依赖 (transitive function dependency)，指对于关系 R 中的属性集合 X, Y, Z

如果有  $Y \not\rightarrow X, Z \not\rightarrow Y, X \rightarrow Y, Y \rightarrow Z, Z \rightarrow X$

称 Z 对 X 传递函数依赖，记作  $X \xrightarrow{\text{trans}} Z \quad (X \xrightarrow{\text{tran}} Y \rightarrow Z)$

注意：Z 不由 Y 决定是为了防止平凡函数依赖的情形

$Y \rightarrow X$  是为了防止，当  $X \rightarrow Y$  且  $Y \rightarrow X$  时，有  $X \leftrightarrow Y$ ，则实际上是  $X \xrightarrow{\text{direct}} Z$

# Database - P21

19 - 10/09 4

逻辑蕴含 (logical implication), 对于函数依赖上定义的蕴含关系

对于关系  $R(u)$ , 如果  $R$  满足函数依赖集合  $F$  中的所有函数依赖  
则关系  $R(u)$  满足另一个函数依赖集合  $G$  中的所有函数依赖

即  $F \rightarrow G$  则称函数依赖  $F$  逻辑蕴含函数依赖  $G$ , 记为  $F \vdash G$

自己是  $F$ :  $\{a\}$  a set of functional dependencies  $L$

logically implies another set of functional dependencies  $T$

即对于任意满足  $F$  函数依赖  $F$  的关系  $R(u)$

如果  $X \rightarrow Y$  是函数依赖集合  $G$  中任意一个函数依赖.

而对于  $R(u)$  中任意两个元组  $s, t$

皆满足  $s[X] = t[X] \rightarrow s[Y] = t[Y]$

则有函数依赖  $F$  逻辑蕴含函数依赖  $G$ .

Armstrong 公理 (Armstrong's axiom), 为了从一组函数依赖求得蕴含的函数依赖的推理规则

对于给定的关系  $R$ , 其属性集合为  $\{r_1, r_2, \dots, r_n\}$ , 而  $X, Y, Z \subseteq \{r_1, r_2, \dots, r_n\}$

自反律 (reflexivity), 指如果有  $Y \subseteq X$ , 则有  $X \rightarrow Y$

证明过程有, 如果有  $Y \subseteq X \subseteq \{r_1, r_2, \dots, r_n\}$

则对于  $R$  中的任意两个元组  $s, t$ , 如果  $s[X] = t[X]$

又  $Y \subseteq X$ , 则有  $s[Y] = t[Y]$

即有如果  $Y \subseteq X$ , 则  $X \rightarrow Y$

增广律 (augmentation), 指如果  $X \rightarrow Y$ , 则  $XZ \rightarrow YZ$ , 注意此处将  $XUZ$  简写为  $XZ$

证明过程有, 如果对于  $R$  中任意两个元组  $s, t$ , 有  $s[XZ] = t[XZ]$

又  $XZ$  表示  $XUZ$ , 则根据自反律

有  $s[X] = t[X]$  且  $s[Z] = t[Z]$

又  $X \rightarrow Y$ , 即有  $s[Y] = t[Y]$

于是有  $s[YZ] = t[YZ]$ , 其中  $YZ$  表示  $YUZ$

即如果  $X \rightarrow Y$ , 则  $XZ \rightarrow YZ$

传递律 (transitivity), 指如果  $X \rightarrow Y$  且  $Y \rightarrow Z$ , 则  $X \rightarrow Z$

证明过程有, 如果对于  $R$  中任意两个元组  $s, t$ , 有  $s[X] = t[X]$

则由于  $X \rightarrow Y$ , 所以有  $s[YZ] = t[YZ]$

再由于  $Y \rightarrow Z$ , 所以有  $s[Z] = t[Z]$

于是有对任意元组  $s, t$ , 如果  $s[X] = t[X]$ , 则有  $s[Z] = t[Z]$

即有  $X \rightarrow Z$

于是有如果  $X \rightarrow Y$  且  $Y \rightarrow Z$ , 则  $X \rightarrow Z$

# Database - P22

Armstrong 公理系统 注意其中自反律可以弱化 (weaken) 为  $X \rightarrow \emptyset$

于是公理系统也可以描述为

$$\vdash X \rightarrow \emptyset$$

] 实际公理 (actual axiom)

$$X \rightarrow Y \vdash XZ \rightarrow YZ$$

] 适当的推理规则

$$X \rightarrow Y, Y \rightarrow Z \vdash X \rightarrow Z$$

] (proper inference rules)

由三条推理规则可以得到更多推理规则

合并规则 (union), 指如果有  $X \rightarrow Y$  且  $X \rightarrow Z$ , 则有  $X \rightarrow YZ$

分离规则 (decomposition), 指如果有  $X \rightarrow YZ$ , 则有  $X \rightarrow Y$  且  $X \rightarrow Z$

注意有时分离规则也描述为

如果有  $X \rightarrow Y$  且  $Z \subseteq Y$ , 则有  $X \rightarrow Z$

伪传递规则 (pseudo-transitivity), 指如果有  $X \rightarrow Y$  且  $WY \rightarrow Z$ , 则有  $WX \rightarrow Z$

注意: 有的合并规则与分离规则可以组合为一个逻辑等价命题

即  $X \rightarrow YZ$  当且仅当  $X \rightarrow Y, X \rightarrow Z$

也称为 分拆 / 组合规则 (splitting / combining)

另外有时也包含 复合规则 (composition)

指如果  $X \rightarrow Y, Z \rightarrow W$ , 则有  $XZ \rightarrow YW$

另外可以对分拆 / 组合规则进行扩展至多个属性集合的情形

即  $X \rightarrow A_1 A_2 \dots A_k$  当且仅当  $\vdash X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_k$

对任意  $A_i (i=1, 2, \dots, k)$ , 有  $X \rightarrow A_i$

闭包 (closure). 指对于关系  $R(u)$  和  $R$  所满足的函数依赖集合  $F$

可由  $F$  中的函数依赖所逻辑蕴含的所有函数依赖构成的集合, 称为  $F$  的闭包, 记为  $F^+$

而对属性集合  $X \subseteq U$ , 即  $X \subseteq \{r_1, r_2, \dots, r_n\}$

是等价的概念

则定义集合  $X_F^+ = \{A \mid X \rightarrow A \text{ 可由函数依赖集合 } F \text{ 导出}\}$ , 注意“蕴含”与“导出”

称属性集合  $X_F^+$  为属性集合  $X$  关于函数依赖集  $F$  的闭包

引理 对于定义在属性集合  $U$  上的函数依赖集合  $F$ , 以及属性集合  $X, Y \subseteq U$

$X \rightarrow Y$  可由  $F$  导出 (即  $X \rightarrow Y \in F^+$ ) 当且仅当  $Y \subseteq X_F^+$

注意这个引理可由扩展的分拆 / 组合规则推出

可以由算法得到定义在属性集合  $U$  上的函数依赖集合  $F$  的闭包  $F^+$

但这个问题属于 NP 完全问题