6 形式化查询语言

Author： 中山大学 17数据科学与计算机学院 YSY

https://github.com/ysyisyourbrother

6.1 关系代数

关系代数是一种*过程化*查询语言

关系代数的基本运算有：*选择、投影、并、集合差、笛卡尔积、更名*

其它运算：*集合交、自然连接、赋值*

6.1.1 基本运算

一元运算：*选择、投影、更名*

二元运算：*并、集合差、笛卡尔积*

6.1.1.1 选择运算

选择(σ)运算选出满足给定谓词的元组

通常，我们允许在选择谓词中进行比较，使用的是：$=, \neq, <, \leq, >, \geq$

另外可以用连词$and(\wedge), or(\vee), not(\neg)$将多个谓词合并为一个较大的谓词

$$\sigma_{dept\_name = \text{"Physics"} \wedge salary > 90\,000}(instructor)$$

6.1.1.2 投影运算

投影(Π)运算返回作为参数的关系

$$\Pi_{ID, name, salary}(instructor)$$

6.1.1.3 关系运算的组合

由于关系代数运算的结果类型仍为**关系**，因此可以把多个关系代数运算组合成一个**关系代数表达式**

$$\Pi_{name}(\sigma_{dept\_name = \text{"Physics"}}(instructor))$$

6.1.1.4 并运算

$$\Pi_{course\_id}(\sigma_{semester = \text{"Fall"} \wedge year = 2009}(section)) \cup \Pi_{course\_id}(\sigma_{semester = \text{"Spring"} \wedge year = 2010}(section))$$

集合运算会自动去除重复值，只留下单个元组。

要使**并**运算$r \cup s$有意义（*相容*），必须满足以下两个条件：

1. 关系r和s必须是同元的，即它们的**属性数目必须相同**
2. 对所有的i，r的第i个属性的域必须和s的第i个属性的**域相同**

6.1.1.5 集合差运算

用$-$表示**集合差**运算表示在一个关系中而不在另一个关系中的那些元组

$$\Pi_{course\_id}(\sigma_{semester="Fall" \wedge year=2009}(section)) - \Pi_{course\_id}(\sigma_{semester="Spring" \wedge year=2010}(section))$$

集合差运算必须在*相容*的关系间进行（与并运算类似），即关系同元且属性的域相同

6.1.1.6 笛卡尔积运算

笛卡尔积(×)运算可以将任意两个关系的信息组合在一起

$$\Pi_{name, course\_id}(\sigma_{instructor.ID=teaches.ID}(\sigma_{dept\_name="Physics"}(instructor \times teaches)))$$

因为可能出现两个关系的属性同名的情况，因此我们把关系名称附加在属性名前：

$$(instructor.ID, instructor.name, instructor.dept\_name, instructor.salary$$
$$teaches.ID, teaches.course\_id, teaches.sec\_id, teaches.semester, teaches.year)$$

可以用选择操作从笛卡尔积中选择出满足特定要求的元组。

6.1.1.7 更名运算

更名(ρ)运算可以给关系赋予名字

我们可以给一个关系更改名字：

$$\rho_x(E)$$

返回表达式E的结果，并把名字x给它。

更名运算的另一形式如下。假设关系代数表达式 E 是 n 元的，则表达式

$$\rho_{x(A_1, A_2, \cdots, A_n)}(E)$$

返回表达式 E 的结果，并赋给它名字 x，同时将各属性更名为 A_1，A_2，…，A_n。

更名运算不是必需的，因为可以对属性使用位置标记。我们可以用位置标记隐含地作为关系的属性名，用 $1、$2、…指代第一个属性、第二个属性，以此类推。位置标记也适用于关系代数表达式运算的结果。下面的关系代数表达式演示了位置标记的用法，我们用它来写先前介绍的计算非最高工资的表达式：

$$\Pi_{\$4}(\sigma_{\$4<\$8}(instructor \times instructor))$$

6.1.3 附加的关系代数运算

6.1.3.1 集合交运算

集合交(∩)运算：$r \cap s = r - (r - s)$

$$\Pi_{course\_id}(\sigma_{semester="Fall" \wedge year=2009}(section)) \cap \Pi_{course\_id}(\sigma_{semester="Spring" \wedge year=2010}(section))$$

6.1.3.2 自然连接运算

r和s的**自然连接**表示为 $r \bowtie s$

$$r \bowtie s = \Pi_{R \cup S}(\sigma_{r.A_1=s.A_1 \wedge r.A_2=s.A_2 \wedge \cdots \wedge r.A_n=s.A_n}(r \times s))$$

其中 $R \cap S = \{A_1, A_2, \cdots, A_n\}$。

6.1.3.3 赋值运算

赋值(←)运算可以给临时关系变量赋值

$r \bowtie s$可以表示为：

$$temp1 \leftarrow R \times S$$
$$temp2 \leftarrow \sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \cdots \wedge r.A_n = s.A_n}(temp1)$$
$$result = \prod_{R \cup S}(temp2)$$

6.1.3.4 外连接运算

外连接运算是连接运算的扩展，它在结果中创建带空值的元组，以此来保留在连接中丢失的那些元组

外连接运算有三种形式：

- 左外连接：$=\bowtie$

  取出左侧关系中所有与右侧关系的任一元组都不匹配的元组，用空值填充所有来自右侧关系的属性
- 右外连接：$\bowtie=$

  与左外连接相反
- 全外连接：$=\bowtie=$

  既做左外连接也做右外连接

左外连接(left outer join)($\overset{\frown}{\bowtie}$)取出左侧关系中所有与右侧关系的任一元组都不匹配的元组，用空值填充所有来自右侧关系的属性，再把产生的元组加到自然连接的结果中。图 6-17 中，元组(58583，Califieri，History，62000，*null*，*null*，*null*，*null*)即是这样的元组。所有来自左侧关系的信息在左外连接结果中都得到保留。

右外连接(right outer join)($\overset{\frown}{\bowtie}$)与左外连接相对称：用空值填充来自右侧关系的所有与左侧关系的任一元组都不匹配的元组，将结果加到自然连接的结果中。图 6-18 中，(58583，*null*，*null*，*null*，*null*，Califieri，History，62000)即是这样的元组。因此，所有来自右侧关系的信息在右外连接结果中都得到保留。

全外连接(full outer join)($\overset{\frown}{\bowtie}$)既做左外连接又做右外连接，既填充左侧关系中与右侧关系的任一元组都不匹配的元组，又填充右侧关系中与左侧关系的任一元组都不匹配的元组，并把结果都加到连接的结果中。

请注意，很有趣的是外连接运算可以用基本关系代数运算表示。例如，左连接运算 $r \overset{\frown}{\bowtie} s$ 可以写成：

$$(r \bowtie s) \cup (r - \prod_R (r \bowtie s)) \times \{(null, \cdots, null)\}$$

其中常数关系$\{(null, \cdots, null)\}$的模式是 $S - R$。

6.1.4 扩展的关系代数运算

6.1.4.1 广义投影

广义投影允许在投影列表中使用算数运算和字符串函数来对投影进行扩展

$$\prod_{ID, name, dept\_name, salary/12}(instructor)$$

例如：

例如，表达式：

$$\prod_{ID, name, dept\_name, salary/12}(instructor)$$

可以得到每个教师的 *ID*、*name*、*dept_name* 以及每月的工资。

6.1.4.2 聚集

聚集函数：输入值的一个汇集，将单一值作为返回值

加了distinct的聚集函数可以去除重复。下面的count-distinct名字相同就只会统计一次

$\mathcal{G}_{count\_distinct(ID)}(\sigma_{semester = "Spring" \wedge year = 2010}(teaches))$　　$_{dept\_name}\mathcal{G}_{average(salary)}(instructor)$

聚集运算$\mathcal{G}$(花体g)通常的形式：　$_{G_1,G_2,\cdots,G_n}\mathcal{G}_{F_1(A_1),F_2(A_2),\cdots,F_m(A_m)}(E)$

G后面的相当于聚集函数avg、sum等，G前面的相当于group by。

6.2 元组关系演算

关系代数表达式产生过程序列，而*元组关系演算*是**非过程化**的，它只需要描述所需信息，而不给出获得该信息的具体过程。

查询表达式为：$\{t|P(t)\}$ 它是所有使谓词p为真的元组集合

6.2.1 查询示例

用这种记法，我们可以将查询"找出工资大于 80 000 美元的所有教师的 *ID*"表述为：

$$\{t \mid \exists s \in instructor(t[ID] = s[ID] \wedge s[salary] > 80000)\}$$

我们可以这样来读上述表达式："它是所有满足如下条件的元组 t 的集合：在关系 *instructor* 中存在元组 s 使 t 和 s 在属性 *ID* 上的值相等，且 s 在属性 *salary* 上的值大于 80 000 美元"。

元组变量 t 只定义在 *ID* 属性上，因为这一属性是对 t 进行限制的条件所涉及的唯一属性。因此，结果得到(*ID*)上的关系。

6.2.2 形式化定义

如果元组变量不被∃或∀修饰，这称为*自由变量*，否则称为*受限变量*

$$t \in instructor \wedge \exists s \in department(t[dept\_name] = s[dept\_name])$$

t 是自由变量，而元组变量 s 称为受限变量。

元组关系演算的公式由*原子*构成，原子可以是一下形式之一：

- $s \in r$：s是元组变量，r是关系
- $s[x]\ \Theta\ u[y]$：其中s和u是元组变量，x和y是属性，Θ是比较运算符（如：<）
- $s[x]\ \Theta\ c$：其中s是元组变量，x是属性，c是常量，Θ是比较运算符

6.2.3 表达式的安全性

P的**域dom(P)**是P所引用的所有值的集合

6.3 域关系演算

关系演算的另一种形式称为**域关系演算**，使用从属性域中取值的域变量，而不是整个元组的值

6.3.1 形式化定义

域关系演算中的表达式形式：$\{ < x_1, x_2, \cdots, x_n > \mid P(x_1, x_2, \cdots, x_n)\}$（$x_n$代表域变量）

域关系演算中的原子具有如下形式之一：

- $< x_1, x_2, \cdots, x_n > \in r$，其中 r 是 n 个属性上的关系，而 $x_1, x_2, \cdots, x_n$ 是域变量或域常量。
- $x \Theta y$，其中 x 和 y 是域变量，而 Θ 是比较运算符（$<$，$\leqslant$，$=$，$\neq$，$>$，$\geqslant$）。我们要求属性 x 和 y 所属域可用 Θ 比较。
- $x \Theta c$，其中 x 是域变量，Θ 是比较运算符，而 c 是 x 作为域变量的那个属性域中的常量。

6.3.2 查询的例子

- 找出在物理系的所有教师姓名，以及他们教授的所有课程的 *course_id*：

$$\{ < n, c > \mid \exists i, a, se, y (< i, c, a, s, y > \in teaches \wedge \exists d, s (< i, n, d, s > \in instructor \wedge d = "Physics")) \}$$

6.4 总结

见P140