ASM语法

Operations

ASM主要支持两种操作：

- **Unitary Operator 单元操作符**
  - count, sum, min, max, average, distinct, not
- **Binary Operator 二元操作符**
  - <, >, <=, >=, =, ==, neq, +, -, *, /, and, or, in, cross, search

注意到此处的"="是**赋值**操作符，细节会稍后解释。

除此之外，ASM还支持两种特殊的数据库操作: **Selection & Projection**

- 对于**Selection**你可以用

$$(\text{Query})\ \text{where}\ (\text{Query})$$

  实现

- 对于**Projection**, 你可以用

$$(\text{Query}).\text{attribute}$$

  实现

此处Query可以是任意形式的，也就是说如下的query

$$((\text{MOVIE cross ROLE})\ \text{where}\ (\text{self.M\_title} = \text{self.ROLE\_title})).\text{M\_year}$$

也是被ASM支持的 (这里演示的是另一个比较重要的数据库操作: **equijoin**)

因此, **我们的ASM是高度灵活的.**

Details

这里介绍一些特殊操作的细节

Selection

对于单个list的selection， 你可以忽略where前list的括号，但是对于复杂list的selection，**你必须在where前的query上加上括号**：

e.g.

$$\begin{aligned}
&\text{MOVIE where (Query)} \quad --- \quad \checkmark \\
&\text{MOVIE where Query} \quad --- \quad \times \\
&\text{MOVIE corss ROLE where (Query)} \quad --- \quad \times
\end{aligned}$$

如果你想调用被选数据的属性，你可以用$self.attribute$访问这些属性。这个self的作用域是在Query里都成立（除了碰到Query里的query，那时内层query的self就是内层query where前的表的数据）

e.g.

$$\text{MOVIE where } ((\text{self.M\_title} = \text{StarWars}) \text{ and } (\text{self.M\_year} = 1977))$$

$$\text{PERSON where } (\text{self.PERSON\_id in } ((\text{ROLE where self.ROLE\_title} = \text{Matrix}).\text{ROLE\_id})$$

前一个$self$代表PERSON的数据，后一个内层query的$self$代表ROLE的数据

cross 笛卡尔积

cross 这里表示的是笛卡尔积, i.e. 合并两个表

e.g.

M_title	M_year	runtime
Matrix	1999	136
StarWar	1977	121

$$\text{cross}$$

ROLE_id	ROLE_title	ROLE_year
1	Matrix	1999
2	3Body	2999

$$=$$

M_title	M_year	runtime	ROLE_id	ROLE_title	ROLE_year
Matrix	1999	136	1	Matrix	1999
Matrix	1999	136	2	3Body	2999
StarWar	1977	121	1	Matrix	1999
StarWar	1977	121	2	3Body	2999

然后就能用**where**实现数据库里常见的**equijoin**

$$(\text{MOVIE cross ROLE}) \text{ where } ((\text{self.M\_title} = \text{self.ROLE\_title}) \text{ and } (\text{self.year} = \text{self}))$$

结果将会是：

M_title	M_year	runtime	ROLE_id	ROLE_title	ROLE_year
Matrix	1999	136	1	Matrix	1999

注意这里的两张表的attributes名必须不同

Projection

主要有两种projection，都用 "." 操作符实现

一种是对单条数据的projection, i.e. self, 就是单条数据的某个属性

另一种是对一张表的projection，会返回一张对应属性的表

e.g.

MOVIE =

M_title	M_year	runtime
Matrix	1999	136
StarWar	1977	121

$$\text{MOVIE.M\_year} \Rightarrow [1999, 1977]$$
$$\text{MOVIE where } (\text{self.M\_year} = 1977) \Rightarrow$$
$$\text{the self.M\_year would be 1999 or 1977 in one loop}$$

请最好在投影操作外再加一层括号，如果你想要继续处理数据的话，因为操作符之间没有优先级。

e.g.

$$(\text{MOVIE where } (\text{runtime} > 100)).\text{M\_year} \quad --- \quad ✓$$
$$1999 \text{ in } (\text{MOVIE where } (\text{runtime} > 100)).\text{M\_year} \quad --- \quad \times$$
$$1999 \text{ in } ((\text{MOVIE where } (\text{runtime} > 100)).\text{M\_year}) \quad --- \quad ✓$$

Search 搜索

这里的search操作会访问数据库的索引结构，以获得比遍历更快的搜索速度

这是一个二元操作符，语法是

$$@\text{A search } \{\text{data:1,year:10}\}$$

这里的字典由attribute和其对应的值构成

search和这样的selection是等价的:

$$\text{A where } ((\text{self.data} = 1) \text{ and } (\text{self.year} = 10))$$

只不过要快很多

一个例子:

$$(\text{A x B}) \text{ where } (\text{self.data in } ((@\text{A search } \{\text{data:1,year:10}\}).\text{data}))$$

注意：因为A在这里是一个关键字（表的名称），如果我们不在A前加上@，其就会输出相对应的表而不是“A”这个名称的string，就会使search失败。所以**请务必在要搜索的表名称前加@**。

Assignment 赋值

我们可以将query的结果储存在变量里，再在之后的query里调用这个变量

语法很简单: $A = Query$

这里A是变量的名称，只要不是关键字（表的名称/操作的名称）就随便取。

注意：变量只能用来存储表格!（指的是存储关系的表格，如果是一个全是id的list，那么是不允许的）

例子:

$$\mathrm{B} = (\mathrm{RESTRICTION\ where\ (self.RESTRICTION\_description} == \mathrm{R18}))$$
$$\mathrm{C} = (\mathrm{B\ cross\ ROLE})$$
$$\mathrm{PERSON\ where\ (not\ (self.PERSON\_id\ in\ (A.ROLE\_id)))}$$

Attention 注意

- 两张要被合并的表的attribute**不能重名**
- 操作符之间没有优先级，所以**请最好在被操作元素上加括号**（如果query出bug，大概率是括号少加了）
- 如果你想要一个常量字符串，**请在字符串前加@**
  - example1: @18 而不是 18, 否则18会被转换成float.
  - example2: @RESTRICTION 而不是 RESTRICTION, 因为后者会输出一个list而不是一个表的名字，不能用在search里