

# 知识图谱数据与查询模型

---

彭鹏

湖南大学信息科学与工程学院

[hnu16pp@hnu.edu.cn](mailto:hnu16pp@hnu.edu.cn)

# 目 录

1. RDF图数据模型与SPARQL查询语言
2. 属性图数据模型与Cypher查询语言
3. TinkerPop图计算框架与Gremlin图遍历语言及遍历机

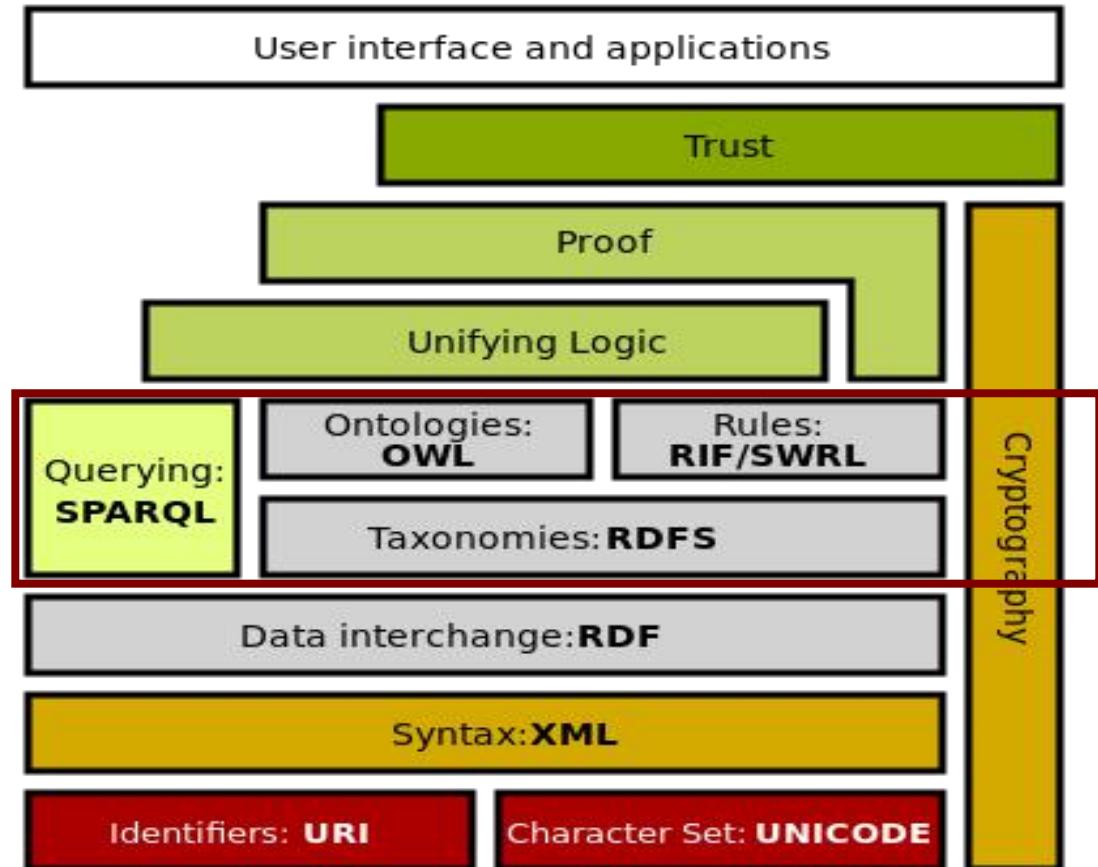
# Part 1

## RDF图数据模型与SPARQL查询语言

# 背景介绍

## 资源描述框架(RDF)数据

1. RDF是知识图谱数据的事实标准
2. RDF是由W3C组织提出的一种描述资源概念模型的语言
3. RDF是语义网的一个基石
4. 语义网的目标是网络上的资源是“机器可理解”(Machine understandable)



## 资源描述框架 (RDF) 发展简史

### 1. 1997 年 10 月，万维网联盟 (W3C) 发布了第一个公开版本 RDF 的草稿

- 由 W3C 一个工作组发布的，该工作组包含了来自 IBM、微软、网景公司 (Netscape)、诺基亚、路透社、SoftQuad 以及美国密歇根大学的代表所组成

### 2. 1999 年，W3C 发布了第一版推荐的 RDF 模型与语法的描述

- 描述了 RDF 的数据模型以及基于 XML 的序列化方法

### 3. 2004 年，RDF 1.0 被提出来

- 提出了关于 RDF 的一组六个描述文档：The RDF Primer、RDF Concepts and Abstract、RDF/XML Syntax Specification (revised)、RDF Semantics、RDF Vocabulary Description Language 1.0 和 The RDF Test Cases

### 4. 2014 年，RDF 1.1 被提出来

- RDF 1.1 包含六个文档：RDF 1.1 Primer、RDF 1.1 Concepts and Abstract Syntax、RDF 1.1 XML Syntax、RDF 1.1 Semantics、RDF Schema 1.1 和 RDF 1.1 Test Cases

# 背景介绍

## 常见RDF图数据库系统

系统名称	系统特点	网址
gStore	北京大学研发的国产RDF 图数据库系统	<a href="http://www.gstore.cn/">http://www.gstore.cn/</a>
Virtuoso	RDF 图数据库系统，用于为DBpedia RDF 数据集提供官方查询接口	<a href="https://virtuoso.openlinksw.com/">https://virtuoso.openlinksw.com/</a>
Jena	惠普实验室开发的开源RDF 图数据库 系统，已经被Apache 基金会管理	<a href="http://jena.apache.org/">http://jena.apache.org/</a>
Blazegraph	RDF 图数据库系统，用于为Wikidata RDF 数据集提供查询接口	<a href="https://blazegraph.com/">https://blazegraph.com/</a>
RDF-3X	德国马克斯·普朗克实验室开发的RDF 图数据库系统	

# RDF图数据模型

## 资源描述框架(RDF)数据示例

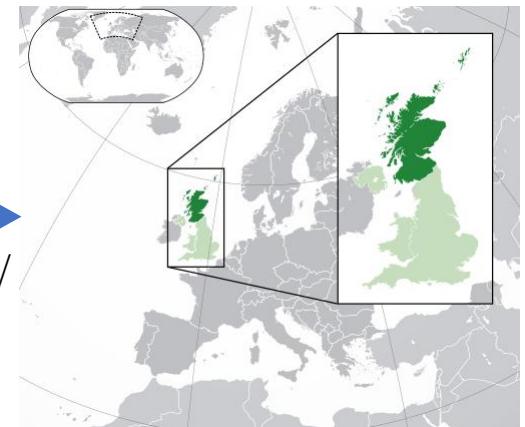
- RDF中任何实体都被称之为资源(Resource), 用一个统一的国际化标识符(IRI, Internationalized Resource Identifier)来唯一标识;
- 资源的属性可以被定义;
- 资源间关系可以被定义;

[http://dbpedia.org/resource/  
University\\_of\\_Glasgow](http://dbpedia.org/resource/University_of_Glasgow)



[http://dbpedia.org/ontology/  
located\\_in](http://dbpedia.org/ontology/located_in)

[http://dbpedia.org/resource/  
Scotland](http://dbpedia.org/resource/Scotland)



# RDF图数据模型

---

## RDF资源

- 现实世界中每个概念、实体和事件都可以对应一个**资源**，可以表示具体的事物也可以是抽象的概念，以及属性；
- 每个资源都用**IRI** (Internationalized Resource Identifier，国际化资源标识符) 进行标识；
- RDF 允许引入不包含任何IRI 标示的资源，被称为**空白结点**或者**匿名资源**，用于标示一种存在变量。空白结点不能用IRI 来全局处理，所以为了区分不同的空白结点，RDF 解析器一般会为每个空白结点分配一个系统生成的内部名。

# RDF图数据模型

## 国际化资源标识符IRI

IRI 是一个用来标识资源的字符串，是数据集中资源的一个唯一的身份ID；当原始的IRI长度过长时，为了方便表达可以引入前缀（prefix）命名空间等方式来简化，以下是常见前缀

前缀	IRI
rdfs:	<a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a>
rdf:	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#">http://www.w3.org/1999/02/22-rdf-syntax-ns#</a>
xsd:	<a href="http://www.w3.org/2001/XMLSchema#">http://www.w3.org/2001/XMLSchema#</a>
sfn:	<a href="http://www.w3.org/ns/sparql#">http://www.w3.org/ns/sparql#</a>
dbr:	<a href="http://dbpedia.org/resource/">http://dbpedia.org/resource/</a>
dbo:	<a href="http://dbpedia.org/ontology/">http://dbpedia.org/ontology/</a>
dbp:	<a href="http://dbpedia.org/property/">http://dbpedia.org/property/</a>

# RDF图数据模型

---

## RDF三元组

每个资源的一个属性及属性值，或者它与其他资源的一条关系，都被表示成<主体，谓词，客体>的**三元组**形式，一个三元组又称为**陈述**

- 所谓**主体**，它是一个资源或者是一个空白节点；
- 所谓**属性/谓词**，是用来描述资源之间的语义关系，或者描述某个资源和属性值之间的关系；
- 所谓**客体**，它可以是一个资源，也可以是一个字面值，也可以是一个空白节点

# RDF图数据模型

## RDF数据集

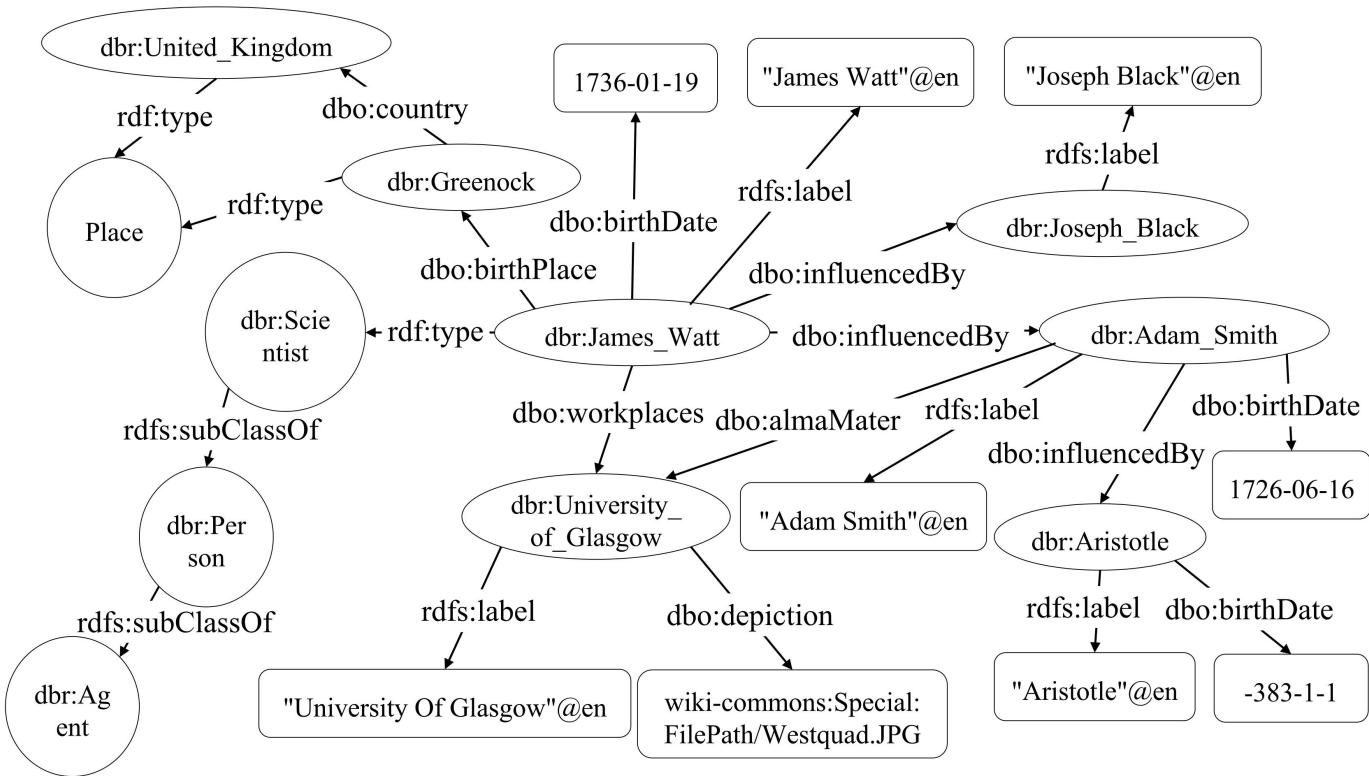
- 给定资源标识符集合 $I$ 、空白节点集合 $B$ 和字面值 $L$ ，一条三元组 $t$ 是属于 $(I \cup B) \times I \times (I \cup B \cup L)$ 的一个元素
- 一个RDF 数据集 $T$ 是 $(I \cup B) \times I \times (I \cup B \cup L)$ 的一个子集

主体	属性	客体
dbr:James_Watt	rdfs:label	"James Watt"@en
dbr:James_Watt	dbo:birthDate	"1736-01-19"^^xsd:date
dbr:James_Watt	dbo:birthPlace	dbr:Greenock
dbr:James_Watt	rdf:type	dbo:Scientist
dbr:James_Watt	dbo:influencedBy	dbr:Joseph_Black
dbr:James_Watt	dbo:influencedBy	dbr:Adam_Smith
dbr:James_Watt	dbp:workplaces	dbr:University_of_Glasgow
dbr:Adam_Smith	rdfs:label	"Adam Smith"@en
dbr:Adam_Smith	dbo:birthDate	"1723-06-16"^^xsd:date
dbr:Adam_Smith	dbo:almaMater	dbr:University_of_Glasgow
dbr:Adam_Smith	dbo:influencedBy	dbr:Aristotle
dbr:Joseph_Black	rdfs:label	"Joseph Black"@en
dbr:University_of_Glasgow	name	"University Of Glasgow"@en
dbr:Aristotle	rdfs:label	"Aristotle"@en
dbr:Aristotle	dbo:birthDate	"-383-1-1"^^xsd:date
dbr:Greenock	dbo:country	dbr:United_Kingdom
dbr:Greenock	dbo:type	dbo:Place
dbr:United_Kingdom	dbo:type	dbo:Place
dbo:Scientist	dbo:subClassOf	dbo:Person
dbo:Person	dbo:subClassOf	dbo:Agent

# RDF图数据模型

## RDF图

- 三元组的主体和客体就是RDF 图中的一系列节点
- 一个谓词的资源标识符在同一张图里可能充当节点，也可能充当边。



# RDF图数据模型

---

## RDF字面值

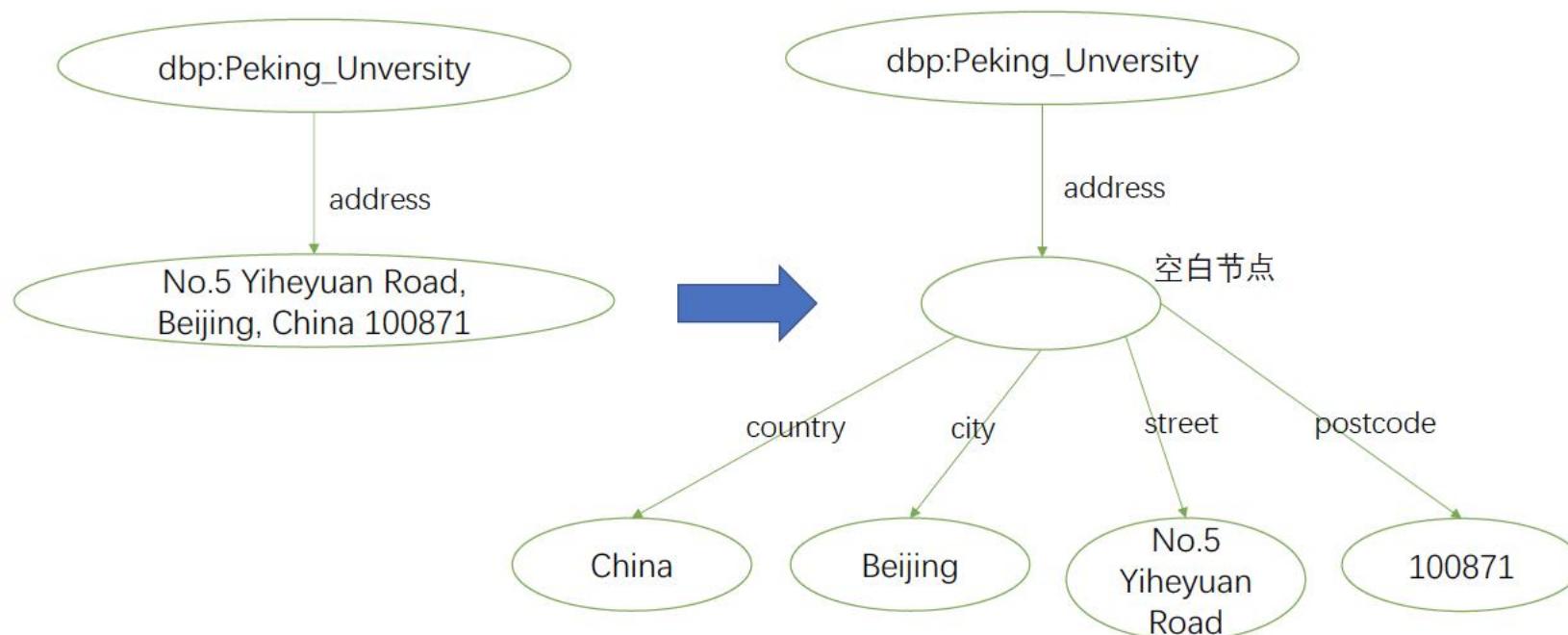
在RDF 的定义中，字面值只会出现在RDF 三元组的客体中。有两种表达方式：

- 朴素文本(plain literals)，也就是普通意义上的字符串
- 类别化文本(typed literals)，可以指定某个字面值的数据类型，参考XML 语法中数据类型定义的[19]，包括xsd:integer，xsd:decimal 等

# RDF图数据模型

## RDF空白节点

**空白节点(Blank Node)** 是指没有统一资源标识符（IRI）同时在知识图谱数据集外部不需要直接访问的节点  
空白节点的引入可以更加方便地表达多元关系和结构化的数据值



# RDF图数据模型

---

## RDF Schema

**RDF Schema (简称RDFS)**，用来表达实体与类别，以及类别之间、以及属性与属性之间、属性的定义域、值域之间的关系

RDF 预定义了一些核心概念和核心属性，这些概念并不提供某个具体领域专用的类别和属性，但是RDFS 为定义某个领域的本体概念提供了基础

# RDF图数据模型

---

## RDF Schema

- **核心类**
  - rdfs:Class: 所有类的类
  - rdfs:Resource: 所有资源的类
  - rdfs:Literal: 所有字面值的类
  - rdfs:Property: 所有属性的类
- **核心属性**
  - rdf:type: 连接一个资源和它属于的类别
  - rdfs:subClassOf: 连接一个类别和它的父类
  - rdfs:subPropertyOf: 连接一个属性和它的子属性
  - rdfs:domain: 定义一个属性的作用域（它的主语的类别）

# RDF图数据模型

## RDF Schema核心类和核心属性举例

三元组	含义
dbp:James_Watt rdf:type dbo:British_Scientist	实体“瓦特”是类别“英国科学家”之一
dbo:British_Scientist rdfs:subclass dbo:Scientist	类别“英国科学家”属于类别“科学家”
dbo:birthPlace rdf:type rdfs:Property	birthPlace 是一个属性
dbo:birthPlace rdfs:domain dbo:Person	birthPlace 的主体一定是类别“人”的实体
dbo:birthPlace rdfs:range dbo:location	birthPlace 的客体一定是类别“地址”的实体

## SPARQL查询语言简介

### SPARQL

- 是W3C 制定的RDF 图数据的标准查询语言；
- SPARQL 从语法上借鉴了SQL，同样属于声明式查询语言；
- SPARQL 提供了强大的基于图匹配的查询功能，也包括可选匹配 (OPTIONAL)、对结果的排序 (ORDER BY)、去重 (DISTINCT) 和限定 (LIMIT) 及值约束条件(FILTER) 等多种操作符，以及直接回答YES/NO 的 ASK 查询等其他形式的查询。同时，SPARQL 语句也具备增、删、改的功能；
- 具体语法细节见SPARQL官网：<https://www.w3.org/TR/rdf-sparql-query/>

# SPARQL查询语言

---

## SPARQL语法

命名空间

```
prefix foaf: <http://xmlns.com/foaf/0.1/>
```

数据集

```
FROM NAMED <http://example.org/foaf/bobFoaf>
```

查询方式

```
SELECT ?name ?bd
```

图模式

```
WHERE { ?p name ?name . ?p birthDate ?bd }
```

结果修饰

```
ORDER BY ?X
```

## SPARQL命名空间

- PREFIX 关键字是把前缀标签和IRI连接起来，一个有前缀的名称是由一个带前缀的标签和一个本地的名称所组成中，其中由冒号“:”来分开
- 常用的命名空间前缀与IRI 的关系之前已经显示了

## SPARQL数据集

- 一个SPARQL 查询可以在包含一个或者多个RDF 数据图的RDF数据集上执行
- 指定使用默认图用关键字FROM，而指定使用某个命名图用关键字FROM NAMED

## SPARQL的六种查询方式

1. **选择查询（SELECT 查询）**， 用来从RDF 数据中选择出满足条件的值来构造一张关系表并返回；
2. **构造查询（CONSTRUCT 查询）**， 用来从RDF 数据中选择出满足条件的值并用这些值按条件构造一个新的RDF 数据；
3. **询问查询（ASK 查询）**， 用来从RDF 数据中判定出RDF 数据中是否有满足条件的值并返回 True/False；
4. **描述查询（DESCRIBE 查询）**， 用来从RDF 数据中选择出一个RDF 数据图来描述某些特点的资源；
5. **插入语句（INSERT 语句）**， 向RDF 数据中插入一个或者多个三元组；
6. **删除语句（DELETE 语句）**， 从RDF 数据中删除一个或者多个三元组。

## SPARQL图模式

- 本章主要介绍选择（SELECT）语句，其中WHERE子句中表示查询语句的正文部分称为**图模式**

## SPARQL三元组模式与基本图模式

- 一个**三元组模式 (Triple Pattern)**  $t$  定义为  $t \in (\mathcal{V} \cup \mathcal{I} \cup \mathcal{B}) \times (\mathcal{V} \cup \mathcal{I}) \times (\mathcal{V} \cup \mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$ ;
- 一个**基本图模式 (Basic Graph Pattern, BGP)** 的定义如下：一条三元组模式  $t$  是一个基本图模式；如果  $P_1$  和  $P_2$  都是基本图模式，则  $P_1 \text{ AND } P_2$  也是一个基本图模式

## SPARQL图模式

- 一个图模式 (Graph Pattern) 是由下面递归定义的：
  1. 如果P 是一个基本图模式，则P 也是一个图模式；
  2. 如果 $P_1$  和 $P_2$  都是图模式，则 $P_1 \text{ AND } P_2$  也是一个图模式；
  3. 如果 $P_1$  和 $P_2$  都是图模式，则 $\{P_1\} \text{ UNION } \{P_2\}$ ,  $P_1 \text{ OPTIONAL } \{P_2\}$  都是图模式。注意到 $\{P_i\}$  表示一个组图模式；
  4. 如果P 是一个图模式，C 是SPARQL 中的FILTER 过滤条件，则 $(P \text{ FILTER } C)$ 也是一个图模式。

## SPARQL组图模式、联合图模式

- 一个**组图模式 (Group Graph Pattern)** 如下递归定义：
  1. 如果 $P$  是一个图模式，则 $\{P\}$  是一个组图模式
  2. 如果 $P$  是一个组图模式，则 $P$  也是一个图模式
- 一个**联合图模式 (Union Graph Pattern)** 如下递归定义：
  1. 如果 $\{P_1\}$  是一个组图模式或者联合图模式， $P_2$  是一个组图模式，则 $(P_1 \text{ UNION } P_2)$  是一个联合图模式
  2. 如果 $(P_1 \text{ UNION } P_2)$  是一个联合图模式，则 $(P_1 \text{ UNION } P_2)$  也是一个图模式。

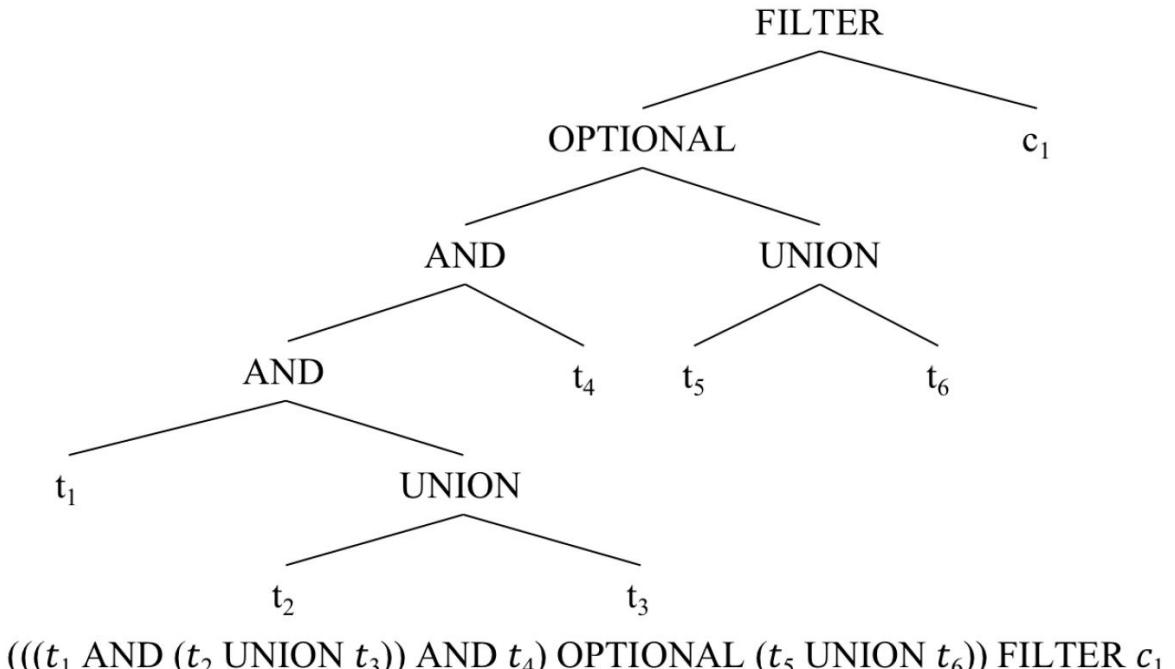
## SPARQL操作符优先级

- 优先级从大到小分别是: (group)、UNION、AND、OPTIONAL、FILTER。
- OPTIONAL 操作符是左结合 (left-associative) 的

# SPARQL查询语言

## SPARQL图模式二叉树

```
SELECT ?x ?name ?birth ?friend WHERE {  
    t1      ?x dbp:workplaces dbr:University_of_Glasgow.  
    t2      {?x dbo:name ?name}  
    UNION  
    t3      {?x rdfs:label ?name}  
    t4      ?x dbo:birthDate ?birth  
    c1      FILTER(?birth = "1736-01-19"^^xsd:date )  
    OPTIONAL{  
        t5          {?x dbo:influencedBy ?friend }  
        UNION  
        t6          {?friend dbo:influencedBy ?x}  
    }  
}
```



## SPARQL结果修饰

1. **排序 (ORDER BY)** , 用来对结果进行排序;

```
SELECT ?x ?label WHERE { ?x rdfs:label ?label } ORDER BY ?label
```

2. **映射 (PROJECTION)** , 用来从结果中取若干变量组成新的结果;

```
SELECT ?label WHERE { ?x rdfs:label ?label }
```

3. **去重 (DISTINCT)** , 用来只返回不同的结果;

```
SELECT DISTINCT ?label WHERE { ?x rdfs:label ?label }
```

4. **偏移 (OFFSET)** , 用来使结果在指定的数量后开始;

```
SELECT ?label WHERE { ?x rdfs:label ?label } OFFSET 2
```

5. **限定结果数量 (LIMIT)** , 用来对结果数量设置了上限

```
SELECT ?label WHERE { ?x rdfs:label ?label } LIMIT 3
```

## 图模式的匹配

- 图模式 $P$  在RDF 数据集 $D$  上的匹配集（表示为 $PI_D$ ）会产生一系列匹配 $\{ \mu_1, \mu_2, \dots, \mu_n \}$
- 一个**匹配**  $\mu: \mathcal{V} \mapsto \mathbb{I} \cup \mathbb{B} \cup \mathbb{L}$ 是从变量集合 $\mathcal{V}$  到 $\mathbb{I} \cup \mathbb{B} \cup \mathbb{L}$ 的一个部分映射函数，这里 $\mathcal{V}$  表示查询中的变量， $\mathbb{I}$ 、 $\mathbb{B}$ 、 $\mathbb{L}$ 分别表示资源标识符集合、空白节点集合和字面值；
- 注意到SPARQL 的查询语义是基于“包(Bag)”的，所以这些匹配当中是有可能有重复的

## 匹配的相互兼容

- 两个匹配 $\mu_1$ 和 $\mu_2$  称为是**相互兼容 (compatible)** 的 (表示为 $\mu_1 \sim \mu_2$ ) 当且仅当如下成立：对所有变量  $v \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$ ，它们的匹配值 $\mu_1(v) = \mu_2(v)$  是相等的，这里 $\text{dom}(\mu)$  表示在匹配  $\mu$  中的变量集合， $\mu_1 \cup \mu_2$  产生一个新的匹配；如果两个匹配  $\mu_1$  和  $\mu_2$  是不兼容的，则表示为 $\mu_1 \not\sim \mu_2$ 。

## 匹配集合的关系操作

1.  $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 | \mu_1 \in \Omega_1 \wedge \mu_2 \in \Omega_2 \wedge \mu_1 \sim \mu_2\}$
2.  $\Omega_1 \cup_{bg} \Omega_2 = \{\mu_1 | \mu_1 \in \Omega_1\} \cup_{bg} \{\mu_2 | \mu_2 \in \Omega_2\}$
3.  $\Omega_1 \setminus \Omega_2 = \{\mu_1 \in \Omega_1 | \forall \mu_2 \in \Omega_2: \mu_1 \not\sim \mu_2\}$

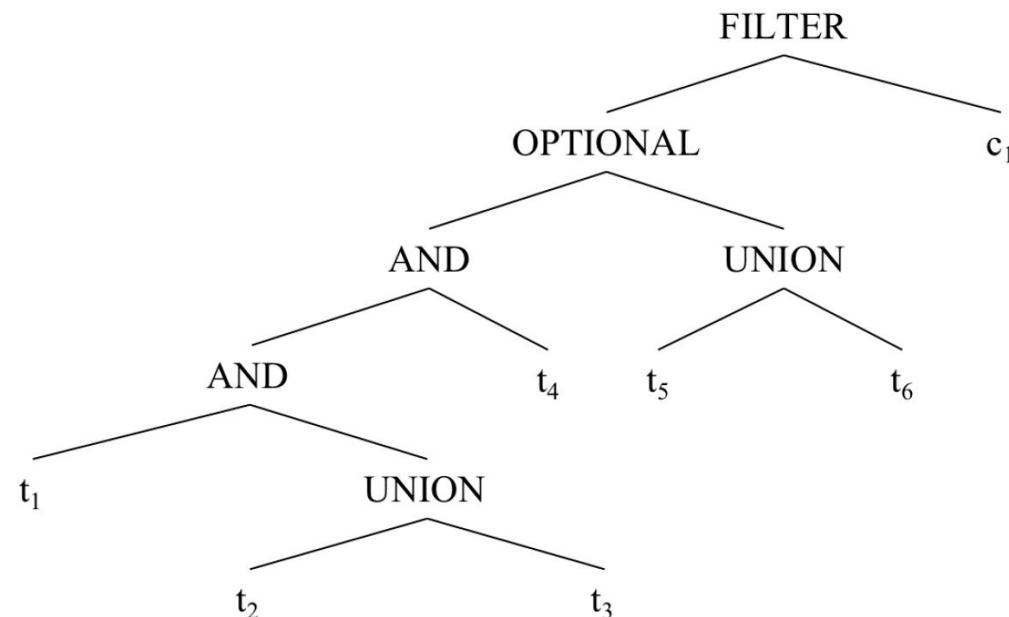
## 图模式在RDF 数据集上的执行结果

图模式 $P$  在RDF 数据集 $D$ 上的执行结果 (表示为 $P|_D$ ) 递归定义如下：

1. 如果 $P$  是一个三元组模式 $t$ ,  $P|_D = \{\mu \mid \text{var}(t) = \text{dm}(t) \wedge \mu(t) \in D\}$ , 其中 $\text{var}(t)$  表示所有 $t$  中间的变量,  $\mu(t)$  表示 $t$  中的变量用匹配 $\mu$  对应的值替代;
2. 如果 $P = (P_1 \text{ AND } P_2)$ , 则 $P|_D = P_1|_D \bowtie P_2|_D$ ;
3. 如果 $P = (P_1 \text{ UNION } P_2)$ , 则 $P|_D = P_1|_D \cup_{bg} P_2|_D$ ;
4. 如果 $P = (P_1 \text{ OPTIONAL } P_2)$ , 则 $P|_D = (P_1|_D \bowtie P_2|_D) \cup_{bg} (P_1|_D \setminus P_2|_D)$ , 其中 $P_1$  称之为 OPTIONAL 的左模式,  $P_2$  称之为 OPTIONAL 的右模式;
5. 如果 $P = (P_1 \text{ FILTER } C)$ , 则 $P|_D = \{\mu \mid \mu \in P_1|_D \wedge \mu(C)\}$  (如果条件 $C$  中的所有变量可以被匹配 $\mu$  对应的值取代 (记做 $\mu(C)$ ) 则 $\mu(C)$  的值就叫做True)

## 图模式的执行

一个图模式P 可以表示为二叉树的形式，所以执行图模式P 查询结果的方法，最为直接的就是基于这颗二叉树采用自底到上的方式来执行。在执行的每一步，利用操作符AND、UNION，OPTIONAL 和FILTER 对其左右儿子的输入



$((t_1 \text{ AND } (t_2 \text{ UNION } t_3)) \text{ AND } t_4) \text{ OPTIONAL } (t_5 \text{ UNION } t_6) \text{ FILTER } c_1$

## SPARQL 1.1的新特性

- 2013 年，SPARQL 1.1被提出，其中引入了一些新特性，包括且不限于：
  - 属性路径（Property Path）
  - 联邦查询（Federated Query）

## SPARQL 1.1的属性路径

- 所谓**属性路径 (Property Path)**，就是找出RDF 数据上两点间满足属性限制条件的路径。这个属性限制条件允许用属性组成的正则表达式来表达

```
SELECT ?x WHERE { dbr:James_Watt dbo:influenceBy* ?x }
```

## SPARQL 1.1的联邦查询

- 所谓**联邦查询 (Federated Query)**，就是允许查询调用远程的其它SPARQL查询接口来与当前的结果进行组合

```
SELECT ?name WHERE {
    dbr:James_Watt dbo:influenceBy ?person .
    SERVICE <https://dbpedia.org/sparql> {
        ?person dbo:name ?name .
    }
}
```

# Part 2

## 属性图数据模型与Cypher查询语言

# 背景介绍

---

## Neo4j背景

- Neo4j 是由Neo4j 公司开发的图数据库系统，其起源于 2000 年Neo4j 的创始人开发的多媒体资产管理系统
- 在这个多媒体资产管理系统中，数据模型经常会发生变化，而且数据结构以及访问控制机制非常复杂。为此， Neo4j 选择了用图模型来存储“关系” 并在此系统中实现了变长的遍历运算
- 该系统通过属性集合的方式来对图上的元素进行标记
- 在Neo4j 中，点、边以及属性都以固定长度记录的形式分别存储在不同的文件中



官网地址：<https://neo4j.com/>

# 背景介绍

---

## Cypher背景

- Cypher 是Neo4j 系统所支持的面向Neo4j所存储属性图的查询语言
- 目前的openCypher项目是一个开发平台，用于Cypher 语言的标准化
- openCypher项目目前形成了一系列成果，包括扩展巴科斯范式、ANTLR4 语法、技术兼容性工具包等等

openCypher

官网地址: <http://www.opencypher.org/>

# 属性图模型

---

## 属性图模型简介

- 属性图模型是一种不同于RDF 三元组的一种图数据模型
- 这个模型由点来表示现实世界中的实体，由边来表示实体与实体之间的关系。同时，点和边上都可以通过键值对的形式被关联上任意数量的属性和属性值
- 在这种图模型中，关系被提到了一个和实体本身一样重要的程度
- 从形式化的角度来看，属性图模型包含三种元素组成：**值、图和表**。

# 属性图模型

## 属性图模型符号列表

值类型	表示单个值的符号	表示值集合的符号
属性键	$k$	$\mathcal{K}$
点标识符	$n$	$\mathcal{N}$
边标识符	$r$	$\mathcal{R}$
点标签	$l$	$\mathcal{L}$
关系标签	$t$	$\mathcal{T}$
函数	$f$	$\mathcal{F}$
值	$v$	$\mathcal{V}$

# 属性图模型

---

## 属性图模型组成元素——值

- 所谓**值**，就是属性图数据模型中涉及的各种数据类型，包括以下这些值类型
  - **标识符**，包括 $N$ 中的点标识符、 $R$ 中的边标识符；
  - **基本数据类型**，包括整数、字符串、布尔值与空值；
  - **链表**，即给定 $m$ 个值 $v_1, v_2, \dots, v_m$ ，链表 $[v_1, v_2, \dots, v_m]$ 也是值；
  - **映射表**，即给定 $m$ 个不同的属性键 $k_1, k_2, \dots, k_m$ 和 $m$ 个值 $v_1, v_2, \dots, v_m$ ，映射表 $\{k_1: v_1, k_2: v_2, \dots, k_m: v_m\}$ 也是值；
  - **路径**，即给定 $m$ 个点 $n_1, n_2, \dots, n_m$ 和 $m - 1$ 条边 $r_1, r_2, \dots, r_{m - 1}$ ，路径 $path(n_1, r_1, n_2, \dots, n_{m - 1}, r_{m - 1}, n_m)$ 也是值，路径经常会被简写成 $n_1r_1n_2\dots n_{m - 1}r_{m - 1}n_m$

# 属性图模型

---

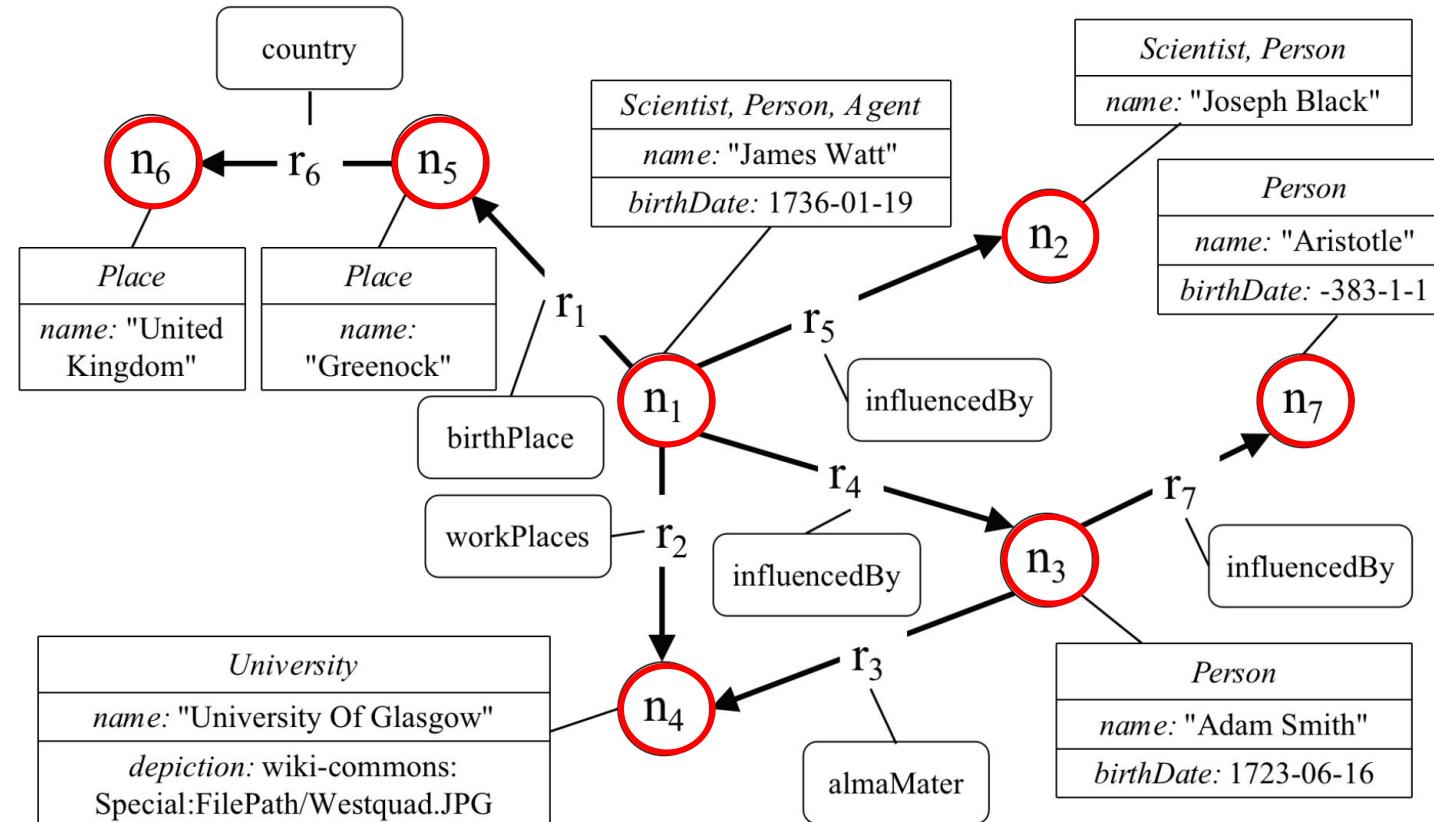
## 属性图模型组成元素——图

- 属性图模型中的图定义七元组  $G = \langle N, R, \text{src}, \text{tgt}, \iota, \lambda, \tau \rangle$ ，
  - $N$  是点标识符集合  $\mathcal{N}$  的子集，表示是图  $G$  中的点集；
  - $R$  是边标识符集合  $\mathcal{R}$  的子集，表示是图  $G$  中的边集；
  - $\text{src} : R \rightarrow N$  是一个函数，表示将一条边映射到其的起点；
  - $\text{tgt} : R \rightarrow N$  是一个函数，表示将一条边映射到其的终点；
  - $\iota : (N \cup R) \times \mathcal{K} \rightarrow V$  是一个函数，表示将一个点或者一条边上一个属性键映射到一个值；
  - $\lambda : N \rightarrow 2^L$  是一个函数，表示将一个点映射到一个有限的点标签集合；
  - $\tau : R \rightarrow T$  是一个函数，表示将一条边映射到一个边类型；

# 属性图模型

## 属性图示例

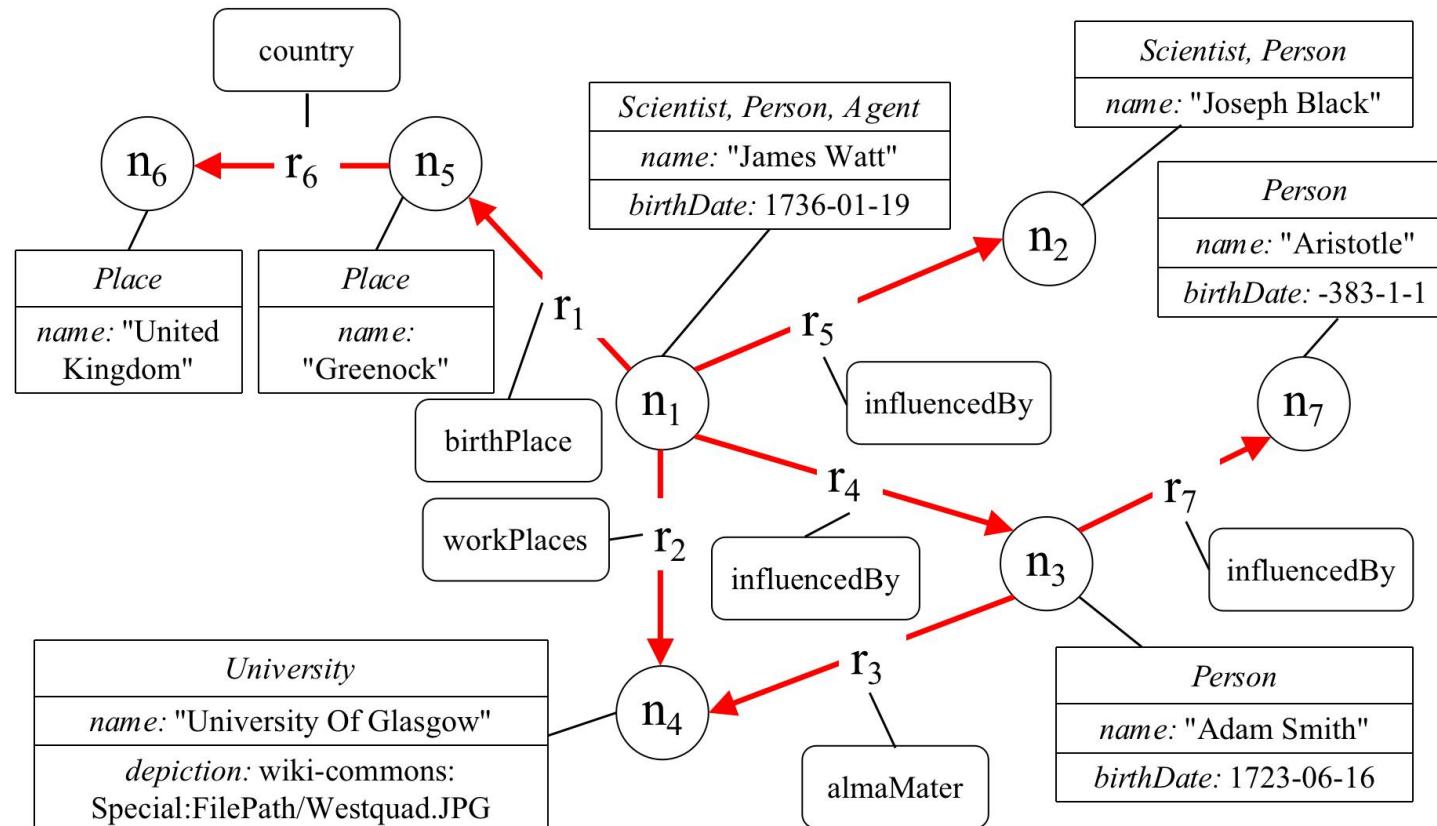
$N = \{n_1, n_2, n_3, n_4, n_5, n_6, n_7\}$  为点集



# 属性图模型

## 属性图示例

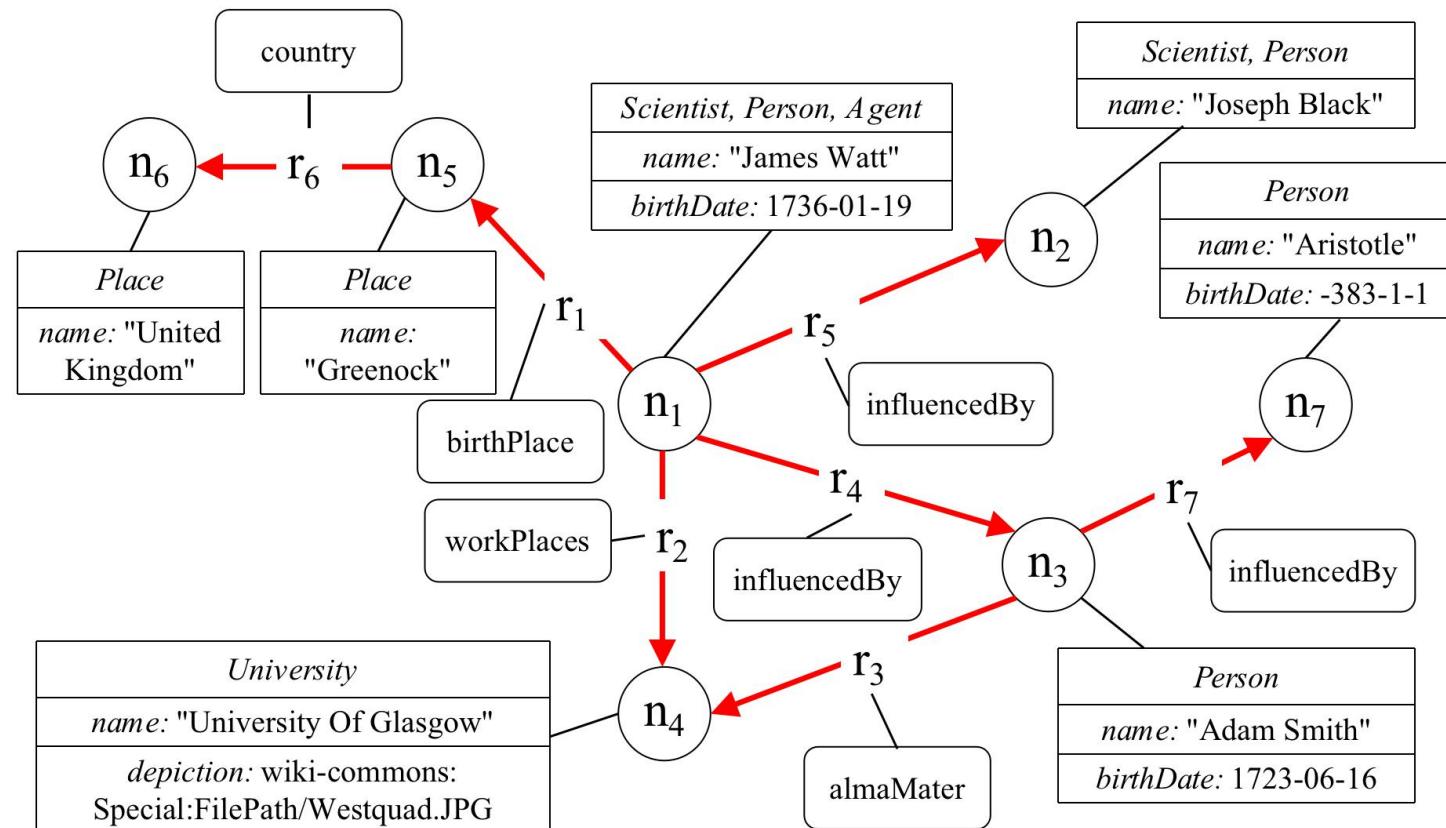
$R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$  为边集



# 属性图模型

## 属性图示例

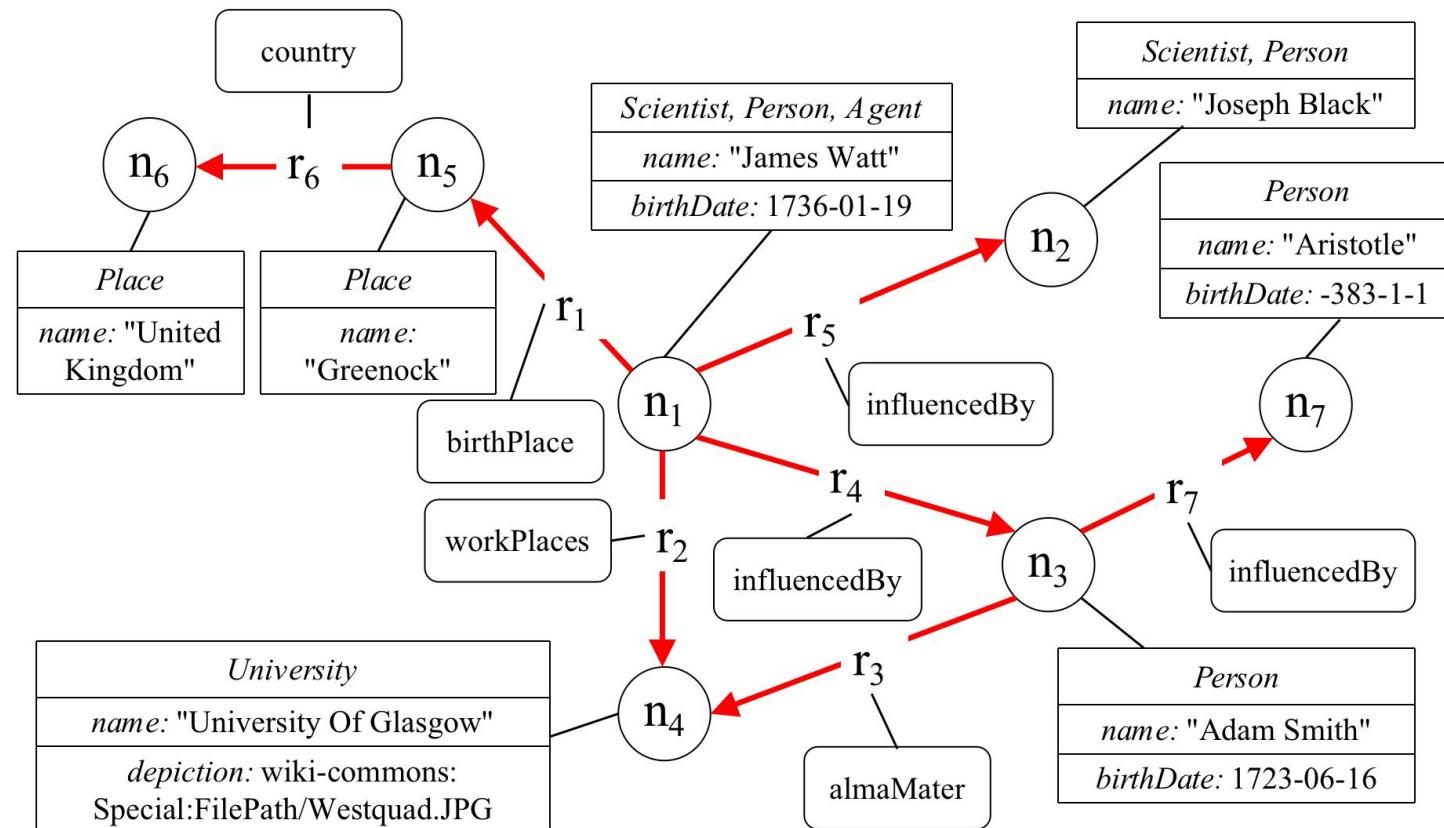
$\text{src} = \{r_1 \rightarrow n_1, r_2 \rightarrow n_1, r_3 \rightarrow n_3, r_4 \rightarrow n_1, r_5 \rightarrow n_1, r_6 \rightarrow n_5, r_7 \rightarrow n_3\}$  用以将各条边映射到其的起点



# 属性图模型

## 属性图示例

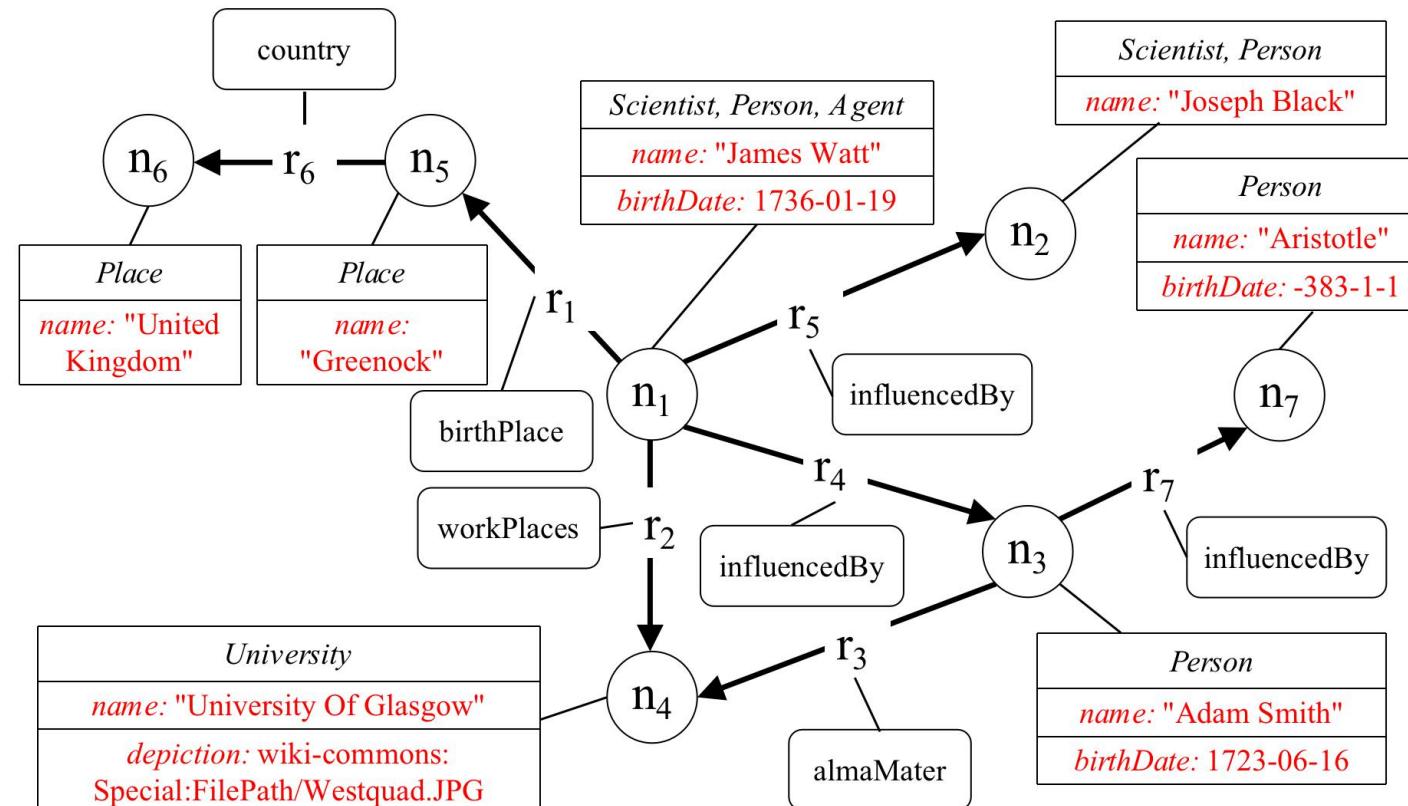
$\text{tgt} = \{r_1 \rightarrow n_5, r_2 \rightarrow n_4, r_3 \rightarrow n_4, r_4 \rightarrow n_3, r_5 \rightarrow n_2, r_6 \rightarrow n_6, r_7 \rightarrow n_7\}$  用以将各条边映射到其的终点



# 属性图模型

## 属性图示例

$\iota(n_1, \text{name}) = \text{"James Watt"}$ ,  $\iota(n_1, \text{birthDate}) = 1736-01-19$ ,  $\iota(n_2, \text{name}) = \text{"Joseph Black"}$ , ...,  $\iota(n_7, \text{birthDate}) = -383-1-1$

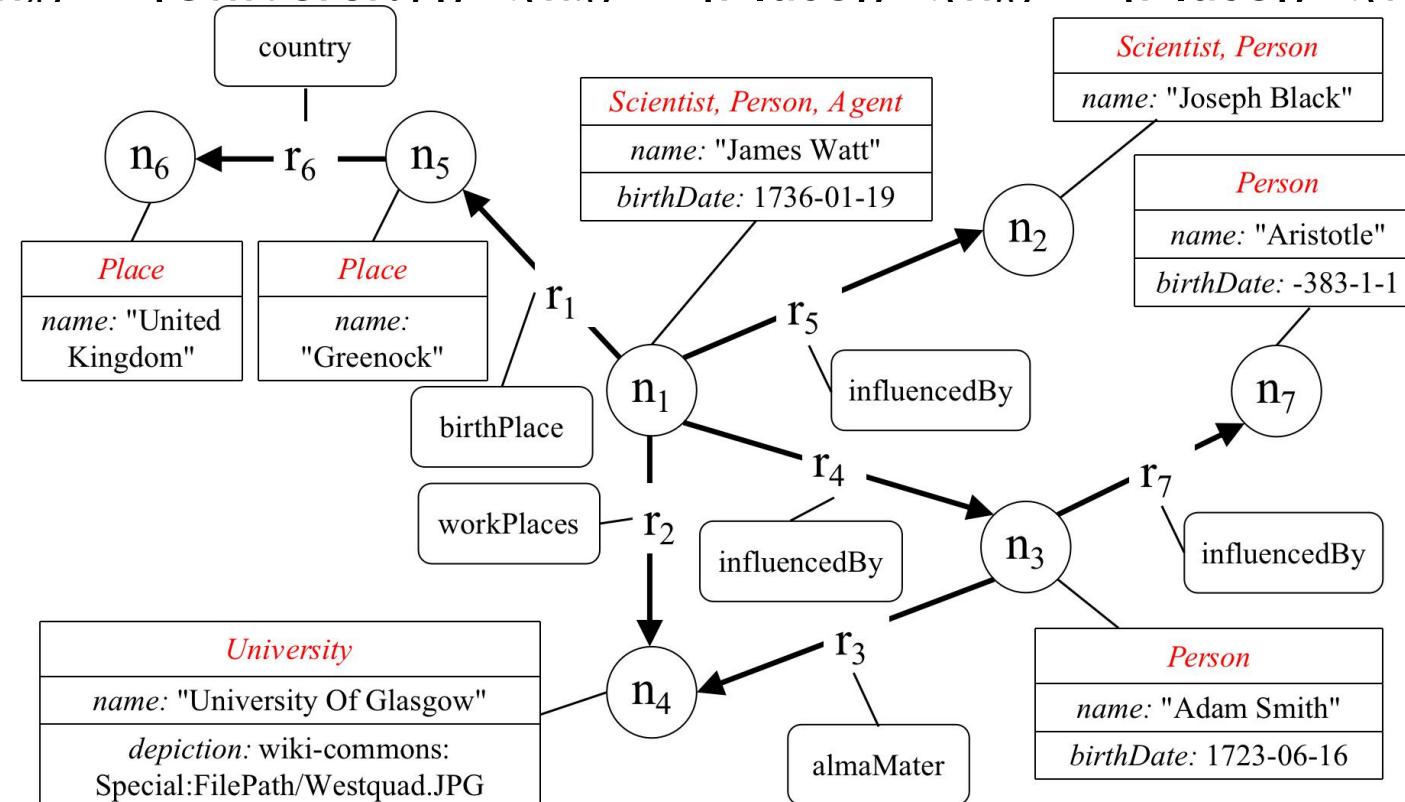


# 属性图模型

## 属性图示例

$\lambda(n_1) = \{\text{Scientist, Person, Agent}\}$ ,  $\lambda(n_2) = \{\text{Scientist, Person}\}$ ,  $\lambda(n_3) = \{\text{Person}\}$

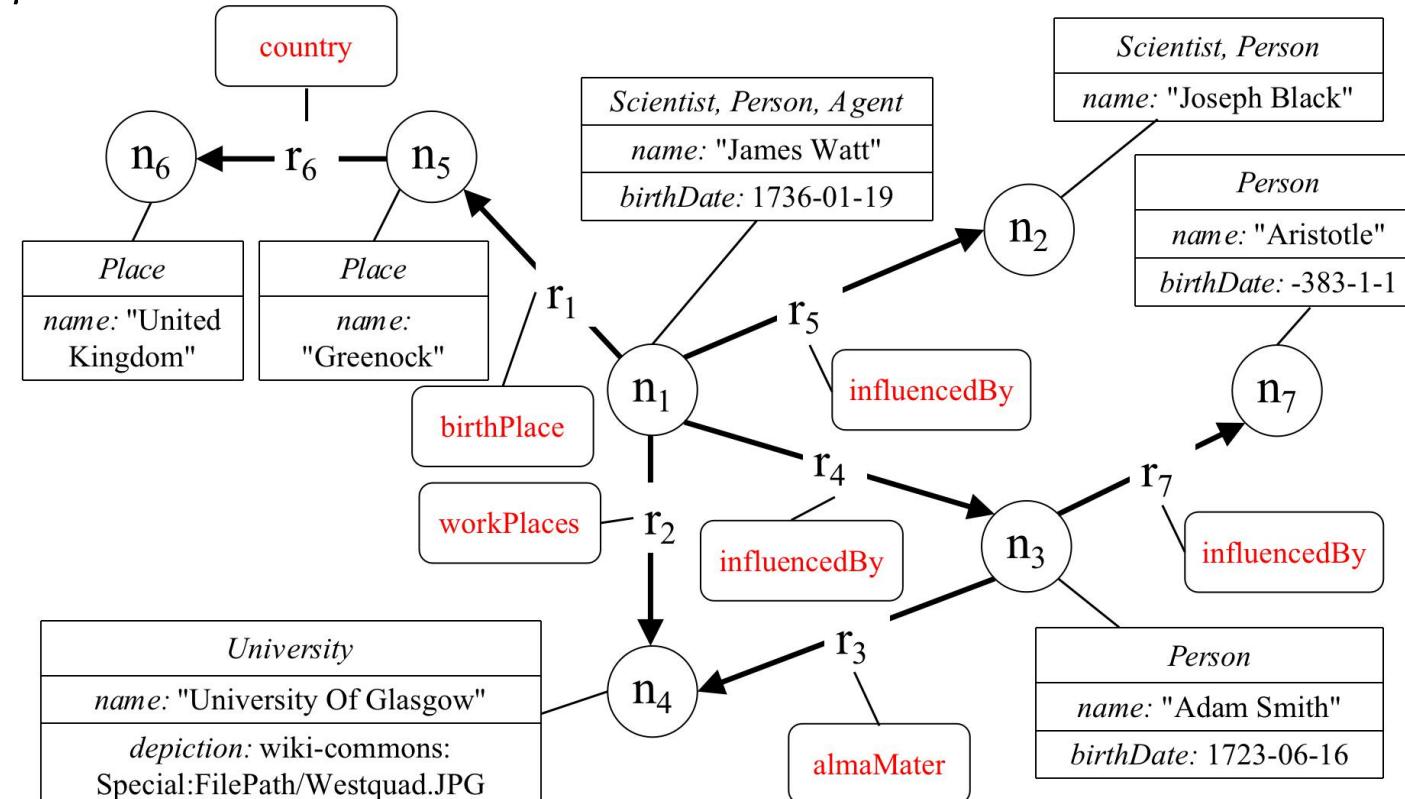
$= \{\text{Person}\}$ ,  $\lambda(n_4) = \{\text{University}\}$ ,  $\lambda(n_5) = \{\text{Place}\}$ ,  $\lambda(n_6) = \{\text{Place}\}$ ,  $\lambda(n_7) = \{\text{Person}\}$



# 属性图模型

## 属性图示例

$\tau(r_1) = \text{birthPlace}$ ,  $\tau(r_2) = \text{workPlaces}$ ,  $\tau(r_3) = \text{almaMaster}$ ,  $\tau(r_4) = \tau(r_5) = \tau(r_7)$   
 $= \text{influencedBy}$ ,  $\tau(r_6) = \text{country}$



# 属性图模型

---

## 属性图模型组成元素——表

- 在属性图模型中，用来定义Cypher 查询语言结果的元素就是表（table）
- 一张表是由多个记录所组成的。给定若干表中的列名 $A = \{a_1, a_2, \dots, a_n\}$ ，一个的记录是一个从 $A$ 到值集合 $V$ 的部分函数
- 一个记录常被记为一个包含列名域的元组 $u = (a_1 : v_1, a_2 : v_2, \dots, a_n : v_n)$ ，其中 $a_1, a_2, \dots, a_n$ 为列名，而 $v_1, v_2, \dots, v_n$ 为值
- 我们将一个记录所涉及的列名集合记为 $\text{dom}(u)$ ，并称之为 $u$  的定义域，它是 $A$ 的子集

# 属性图模型

---

## 属性图模型组成元素——表

- 给定两个定义域完全不相交的记录元组 $u_1 = (a_1:v_1, a_2:v_2, \dots, a_n:v_n)$ 和 $u_2 = (a'_1:v'_1, a'_2:v'_2, \dots, a'_n:v'_n)$ ,  $u_1$  和 $u_2$  的**连接**表示为:

$$(u_1, u_2) = (a_1:v_1, a_2:v_2, \dots, a_n:v_n, a'_1:v'_1, a'_2:v'_2, \dots, a'_n:v'_n)$$

# 属性图模型

---

## 属性图模型组成元素——表

- 在记录的基础上，一张在列名集合 $\Lambda$ 上的表 $T$ 被定义为一个记录的包，这个包中每个记录 $u$ 的定义域是都是 $\Lambda$ ；
- 一张没有记录的空表被表示为 $()$ ；
- 两张表 $T_1$ 和 $T_2$ 的全连接记为 $T_1 \times T_2$ ，也就是将 $T_1$ 中任意一个记录和 $T_2$ 中任意一个记录进行连接所形成的记录集合。

# Cypher查询语言

---

## Cypher语言查询语义

- Cypher语言中一个在图G上的查询Q可以认为是从空表到结果关系表的转换

$$() \xrightarrow{Q|_G} T$$

# Cypher查询语言

---

## Cypher语言查询示例

- 找到属性图中人物的出生地点以及受多少人影响（包括直接和间接影响）

t<sub>1</sub>   **MATCH** (r:Person)

t<sub>2</sub>   **OPTIONAL MATCH** (r) - [:birthPlace] -> (pl:Place)

t<sub>3</sub>   **WITH** r, pl **AS** birthPlace

t<sub>4</sub>   **MATCH** (r) - [:influencedBy\*] -> (p:Person)

t<sub>5</sub>   **RETURN** r.name, birthPlace.name, count(p) **AS** influenceNum

# Cypher查询语言

## Cypher语言查询示例

- 在例子t<sub>1</sub>行的第一个MATCH 子句中的图模式是一个“点模式”，该语句是为了找到点标签为Person 的点，将这些节点的标识符(ID) 赋值给变量r，进而空表转换为以下的表T<sub>1</sub>

```
t1 MATCH (r:Person)
t2 OPTIONAL MATCH (r) - [:birthPlace] -> (pl:Place)
t3 WITH r, pl AS birthPlace
t4 MATCH (r) - [:influencedBy*] -> (p:Person)
t5 RETURN r.name, birthPlace.name, count(p) AS influenceNum
```

r
n <sub>1</sub>
n <sub>2</sub>
n <sub>3</sub>
n <sub>7</sub>

空表 → t<sub>1</sub>

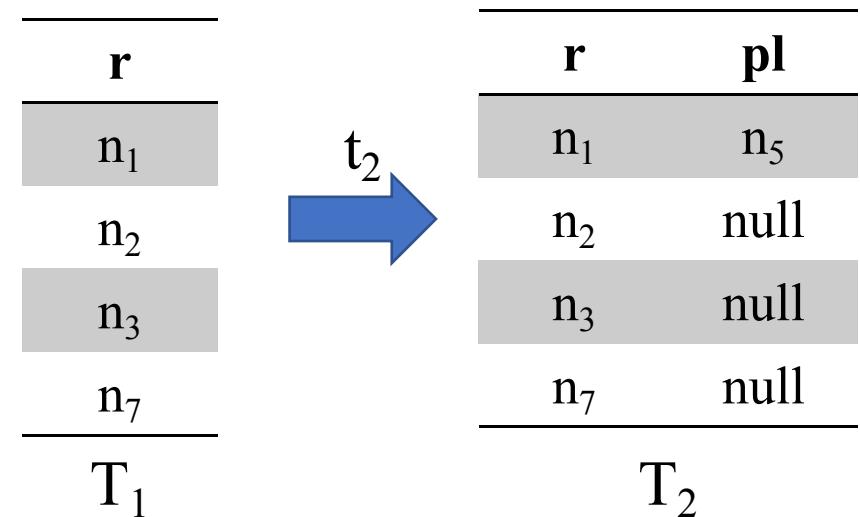
T<sub>1</sub>

# Cypher查询语言

## Cypher语言查询示例

- MATCH 子句也可以通过OPTIONAL 关键词实现类似于SQL 中的左外连接功能
- 比如查询t<sub>2</sub> 行的OPTIONAL MATCH 子句会匹配所有人物的出生地点，然后与查询t<sub>1</sub> 行的结果进行左外连接

```
t1 MATCH (r:Person)  
t2 OPTIONAL MATCH (r) - [:birthPlace] -> (pl:Place)  
t3 WITH r, pl AS birthPlace  
t4 MATCH (r) - [:influencedBy*] -> (p:Person)  
t5 RETURN r.name, birthPlace.name, count(p) AS influenceNum
```



# Cypher查询语言

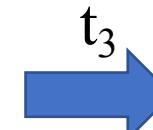
## Cypher语言查询示例

- WITH 子句用在不同子句之间过渡。经过WITH 子句之后，其前面的结果表将会通过映射操作形成新的表，这个新的表会作为下一个子句的输入
- 这一点类似于关系代数中的投影和更名操作
- 比如，查询t<sub>3</sub>行的WITH对变量r和pl进行映射，并将列名pl 重命名为birthPlace

```
t1 MATCH (r:Person)
t2 OPTIONAL MATCH (r) - [:birthPlace] -> (pl:Place)
t3 WITH r, pl AS birthPlace
t4 MATCH (r) - [:influencedBy*] -> (p:Person)
t5 RETURN r.name, birthPlace.name, count(p) AS influenceNum
```

r	pl	r	birthPlace
n <sub>1</sub>	n <sub>5</sub>	n <sub>1</sub>	n <sub>5</sub>
n <sub>2</sub>	null	n <sub>2</sub>	null
n <sub>3</sub>	null	n <sub>3</sub>	null
n <sub>7</sub>	null	n <sub>7</sub>	null

T<sub>2</sub>



T<sub>3</sub> 59

# Cypher查询语言

## Cypher语言查询示例

- Cypher 支持在图查询模式中存在变长的关系模式
- 比如，查询t<sub>4</sub>行的MATCH 子句就包含了一个变长的关系模式influncedBy，这表示在计算这个图模式匹配的时候匹配一个或者多个连续的边，它们的边类型均为influncedBy

```
t1 MATCH (r:Person)
t2 OPTIONAL MATCH (r) - [:birthPlace] -> (pl:Place)
t3 WITH r, pl AS birthPlace
t4 MATCH (r) - [:influncedBy*] -> (p:Person)
t5 RETURN r.name, birthPlace.name, count(p) AS influenceNum
```

r	birthPlace	p
n <sub>1</sub>	n <sub>5</sub>	n <sub>2</sub>
n <sub>2</sub>	null	n <sub>5</sub>
n <sub>3</sub>	null	n <sub>3</sub>
n <sub>7</sub>	null	n <sub>7</sub>

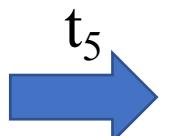
T<sub>3</sub> → T<sub>4</sub>

# Cypher查询语言

## Cypher语言查询示例

- RETURN 语句功能和WITH 子句类似，也是用来对前面对表进行映射操作，只不过最终得到的表会直接返回给用户，这一点等同于SQL 中的SELECT
- 查询 $t_5$  行的RETURN 子句执行完会得到最终结果返回给用户，查询 $t_5$  行RETURN 语句包含一个聚集函数count()，类似于SQL 中的GROUP BY 引导聚集查询

r	birthPlace	p
n <sub>1</sub>	n <sub>5</sub>	n <sub>2</sub>
n <sub>1</sub>	n <sub>5</sub>	n <sub>3</sub>
n <sub>1</sub>	n <sub>5</sub>	n <sub>7</sub>
n <sub>3</sub>	null	n <sub>7</sub>



r.name	birthPlace.name	influenceNum
"James Watt"	"Greenock"	3
"Adam Smith"	null	1

T<sub>5</sub>

# Cypher查询语言

---

## Cypher 查询语言——语法和语义

- 一个Cypher 查询语言包括四部分：表达式、图模式、子句和查询
- 针对一个属性图而言，Cypher 语句既包括查询也包括数据更新和操作等功能

# Cypher查询语言

---

## Cypher 查询语言语法

- 所谓**表达式**，就是出现在Cypher 查询语句中的对于值的操作表达式，包括数值操作、字符串操作、关系操作、链表操作、映射表操作、路径操作等等

$\text{expr} ::= v \mid a \mid f(\text{expr\_list}) \quad v \in \mathcal{V}, a \in \mathcal{A}, f \in \mathcal{F}$

$\mid \text{expr}.k \mid \{\} \mid \{ \text{prop\_list} \}$

$\mid [] \mid [\text{expr\_list}] \mid \text{expr IN expr} \mid \text{expr}[\text{expr}] \mid \text{expr}[\text{expr}..] \mid \text{expr}[\dots\text{expr}] \mid \text{expr}[\text{expr}..\text{expr}]$

$\mid \text{expr STARTS WITH expr} \mid \text{expr ENDS WITH expr} \mid \text{expr CONTAINS expr}$

$\mid \text{expr OR expr} \mid \text{expr AND expr} \mid \text{expr XOR expr} \mid \text{NOT expr} \mid \text{expr IS NULL} \mid \text{expr IS NOT NULL}$

$\mid \text{expr} < \text{expr} \mid \text{expr} \leq \text{expr} \mid \text{expr} \geq \text{expr} \mid \text{expr} > \text{expr} \mid \text{expr} = \text{expr} \mid \text{expr} \neq \text{expr}$

$\text{expr\_list} ::= \text{expr} \mid \text{expr}, \text{expr\_list}$

# Cypher查询语言

---

## Cypher 查询语言语法

- 所谓**图模式**，就是出现在Cypher 查询语句MATCH 子句中查询模式，这个是Cypher 语言的核心
- Cypher 定义了两种基本查询模式：点模式和关系模式

# Cypher查询语言

---

## Cypher 查询语言语法

- **点模式**是一个三元组( $a, L, P$ )，其中 $a$  为所查询点对应的变量名、 $L$  是所查询的点需要满足的标签集合、 $P$  是所查询的点在属性上所需满足的条件
- 点模式的查询语义是找到属性图中的所有满足如下条件的节点：
  1. 这些节点的标签满足 $L$  中的定义；
  2. 这些节点在属性上满足 $P$ 中定义的属性约束条件；并且把这些节点的标识符赋值给变量 $a$

# Cypher查询语言

---

## Cypher 查询语言语法——点模式示例

(r:Person {name: "James Watt"})

- 其中：
  1. 变量r 的标签是Person；
  2. 变量r 在name 属性上要是“James Watt”。

# Cypher查询语言

---

## Cypher 查询语言语法

- **关系模式**是一个五元组( $d, a, T, P, l$ )，其中 $d$  为关系模式的方向、 $a$  是所查询关系对应的变量名、 $T$  是所查询关系需要满足的边类型集合、 $P$  是所查询关系在边属性上所需满足的条件、 $l$  表示所查询关系的可以对应多少条边
- 关系模式的查询语义是找到属性图中的满足如下条件的关系边：
  1. 边的方向符合 $d$  的定义，其中MATCH 语句中 $-[]->$ 、 $<-[]-$ 、 $-[]-$ 分别表示自左向右、自右向左的有向边和无向边；
  2. 边的关系类型满足 $T$  中约束；
  3. 边在属性上满足 $P$  中定义的属性约束条件；
  4.  $l$  表示关系边的数量范围。其中 $[m, n]$  表示边的数量范围在 $m$  和 $n$ 之间

# Cypher查询语言

---

## Cypher 查询语言语法——关系模式示例

( r : Person ) - [:influencedBy \* 1..3] -> ( p : Person)

- 其中：
  1. 边的方向d 此处为自左向右的有向边；
  2. 边的关系类型满足T 中约束，此处为influencedBy；
  3. 边在属性上满足P 中定义的属性约束条件，此处为空；
  4. | 表示关系边的数量范围，此处是[1, 3]。

# Cypher查询语言

## Cypher 查询语言语法

- 图模式的形式化语法如下

pattern ::= pattern<sup>o</sup> | a = pattern<sup>o</sup>

pattern<sup>o</sup> ::= node\_pattern | node\_pattern rel\_pattern pattern<sup>o</sup>

node\_pattern ::= ( a? label\_list? map? )

rel\_pattern ::= - [ a? type\_list? len? map? ] ->  
| <- [ a? type\_list? len? map? ] -  
| - [ a? type\_list? len? map? ] -

label\_list ::= :t | :t label\_list

map ::= { prop\_list }

prop\_list ::= k : expr | k : expr, prop\_list

type\_list ::= :t | type\_list | t

len ::= \* | \*d | \*d<sub>1</sub> .. | \*..d<sub>2</sub> | \*d<sub>1</sub> .. d<sub>2</sub>

d, d<sub>1</sub>, d<sub>2</sub> ∈ N

# Cypher查询语言

---

## Cypher 查询语句语法

- 所谓**子句**，就是表示一个从表到表到函数
- Cypher 中最主要子句的就是MATCH 子句。MATCH 子句后面跟的就是图模式，可以是点模式，也可以是点模式加关系模式，以及由点模式和关系模式组成的路径模式
- MATCH 子句本身的含义是图模式P 在属性图上的匹配的结果关系表T；如果MATCH 子句前面有关键词“OPTIONAL”，表示的是左外连接；WITH 子句是对查询中间结果的一个映射，同时WITH 子句也可以表达聚集函数的操作；WHERE 子句可以用在MATCH 和WITH 子句中，用于对匹配结果进行过滤

# Cypher查询语言

---

## Cypher 查询语言语法

- 子句的形式化语法如下

```
clause ::= [OPTIONAL] MATCH pattern_tuple [WHERE expr]
          | WITH ret [WHERE expr] | UNWIND expr AS a      a ∈ A
pattern_tuple ::= pattern | pattern, pattern_tuple
```

# Cypher查询语言

---

## Cypher 查询语言语法

- 所谓**查询**，就是若干子句组合并以RETURN语句结尾或者多个查询的并

```
query ::= query° | query UNION query | query UNION ALL query
query° ::= RETURN ret | clause query°
ret ::= * | expr [AS a] | | ret , expr [AS a]
```

# Part 3

## TinkerPop图计算框架与 Gremlin图遍历语言及遍历机

## TinkerPop图计算框架

- Apache TinkerPop 是一个由Apache 软件基金会维护的独立于具体图数据库厂商的开源图计算框架
- 如果一个图数据库系统使用图数据作为底层数据模型并能支持使用Gremlin 图遍历语言对其进行查询，那么该系统被称为“**支持TinkerPop 框架的**”  
**(TinkerPop-enabled)**

# 背景介绍

## 常见“支持TinkerPop 框架的”数据系统

系统名称	系统特点
Blazegraph	RDF 图数据库系统
Neo4j	属性图数据库系统
JanusGraph	属性图数据库系统
Titan	属性图数据库系统
阿里巴巴图数据库	属性图数据库系统
Hadoop (Spark)	图计算系统
华为图引擎服务	图计算系统

## Gremlin 图遍历语言与图遍历机

- TinkerPop 框架的核心是Gremlin 图遍历语言
- **Gremlin 图遍历语言**是一种函数式语言，用来定义Gremlin 图遍历机对图数据进行基于遍历的查询
- **Gremlin 图遍历机**是一种逻辑上的自动机。该自动机由若干指令集组成并由执行引擎来执行，执行引擎具体实现独立于编程语言，只要求这个编程语言是支持函数式编程的。目前已经有基于Java、Scala 等多种函数编程语言的执行引擎

## 属性图模型

- 属性图模型定义与Neo4j 图数据库系统及Cypher 查询语言所使用的属性图模型略有不一样
- Gremlin 中的图并没有要求点和边一定要有标签
- 在Gremlin 中，图被定义为三元组 $(V, E, \lambda)$ 。其中， $V$ 是点集合， $E = V \times V$ 是边集， $\lambda: ((V \cup E) \times \Sigma^*) \rightarrow (U \setminus (V \cup E))$  是将点和边及其属性映射到属性值的函数。这里， $\Sigma$ 是字符集，而 $\Sigma^*$ 表示字符串； $U$ 是全集，任何对象都是可以属于全集 $U$ 。 $\lambda$  表示将图上点和边的属性（这个属性就是对应 $\Sigma^*$ 中一个字符串）映射到一个点和边以外 $U$  中任意对象作为其属性值

## JanusGraph背景

- 本书使用JanusGraph作为典型“支持TinkerPop 框架的”数据系统来介绍Gremlin图遍历语言与图遍历机
- JanusGraph 是一个Linux 基金会下的开源分布式图数据库系统并提供Apache2.0 软件许可证
- 这里需要注意，在Jeansgraph里面，每个节点只有一个label 和label 属性值，和其他的是用户自定义的key—value 形式的属性和属性值。

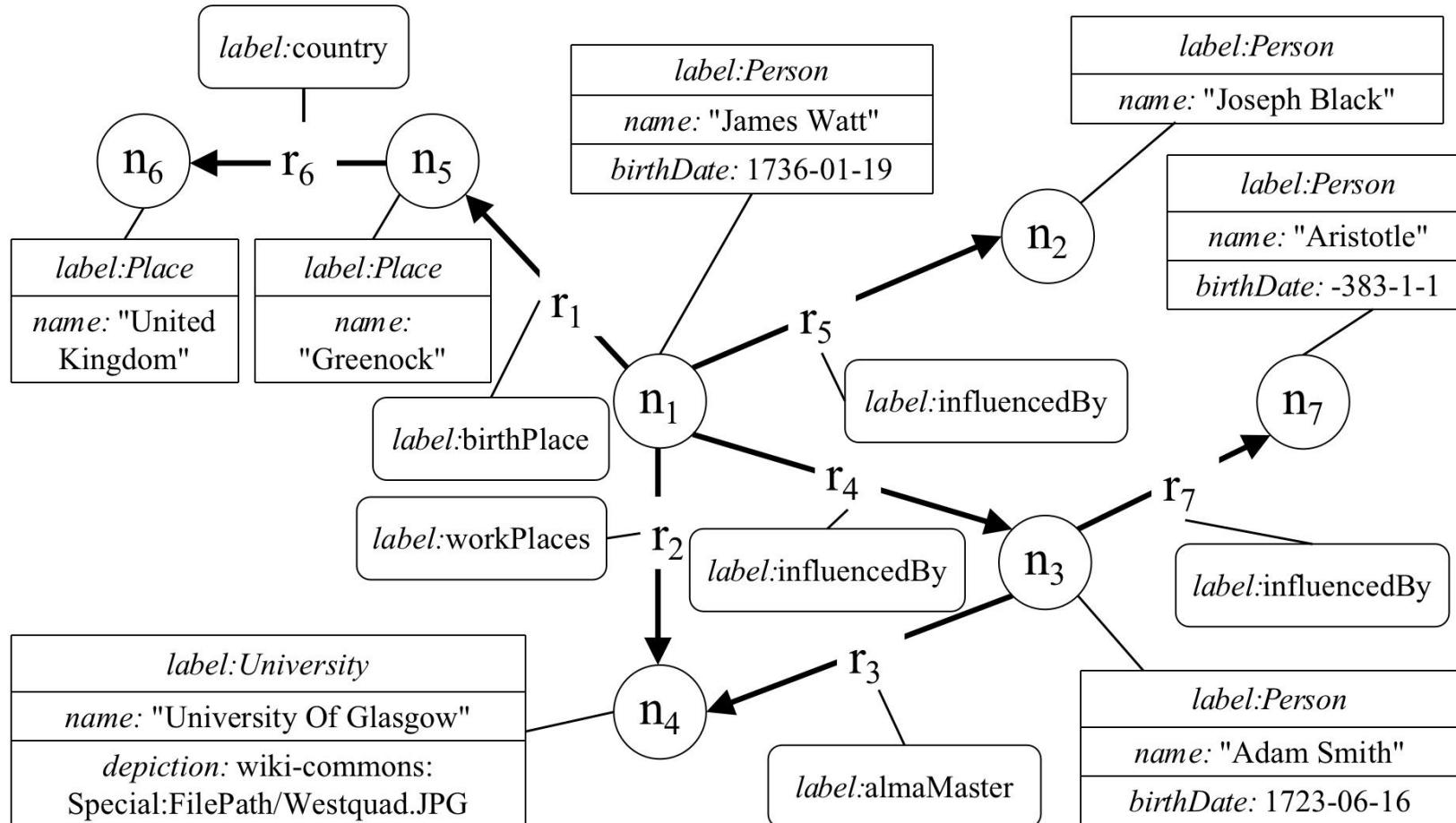


JanusGraph

官网地址: <https://janusgraph.org/>

# 背景知识

## JanusGraph属性图示例



# Gremlin图遍历语言

---

## Gremlin 图遍历语言简介

- Gremlin 图遍历语言是一种逻辑上的编程语言，所以它的具体实现独立于具体编程语言。只需要一个编程语言是支持函数组合和函数作为对象的函数式编程语言，都可以作为实现Gremlin 的**宿主语言**
- **函数组合**可以用来支持链式步骤组合，比如 $f \circ g \circ h$ 在面向对象编程语言中可以通过函数组合表示成 $f().g().h()$
- **函数作为数据类型**用来支持步骤递归式步骤，比如 $f(g \circ h)$ 可以表示成 $f(g().h())$

# Gremlin图遍历语言

---

## 遍历器以及步骤

- Gremlin 以简洁的方式定义一些游标（在Gremlin中被称为**遍历器**）以及这些遍历器在属性图上进行将进行的遍历操作
- 在Gremlin 中，每个遍历操作可由多个函数组成，每一个函数也被称为**步骤**。

## 遍历分类

- 简单遍历
- 分支遍历
- 路径遍历
- 映射遍历
- 变异遍历
- 声明式遍历
- 定义域限定式遍历

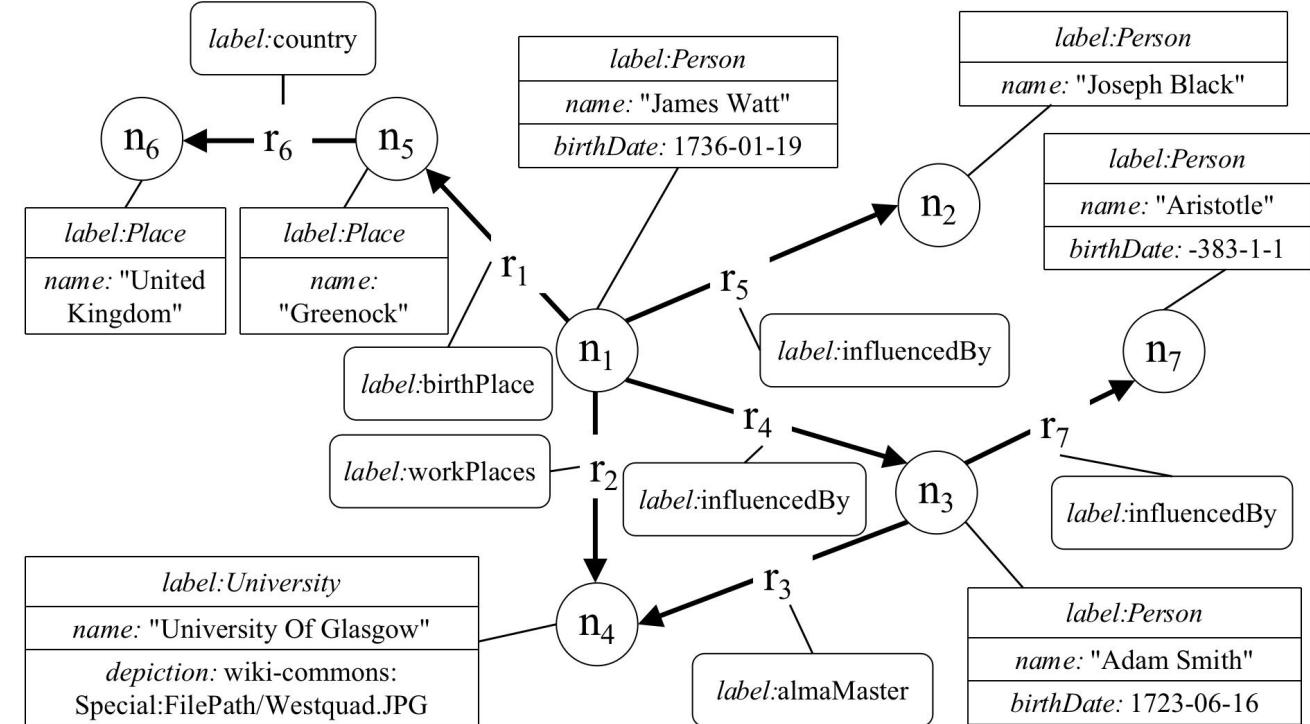
## 简单遍历

- 对于Gremlin 图遍历机而言，最基本遍历操作就是将遍历器沿着一个遍历序列 $\psi_1 \rightsquigarrow \psi_2 \rightsquigarrow \dots \psi_n$ 依次执行，而且这些遍历操作中没有分支与递归
- Gremlin 图遍历语言可以直接用函数的链式组合来实现

# Gremlin图遍历语言

## 简单遍历示例

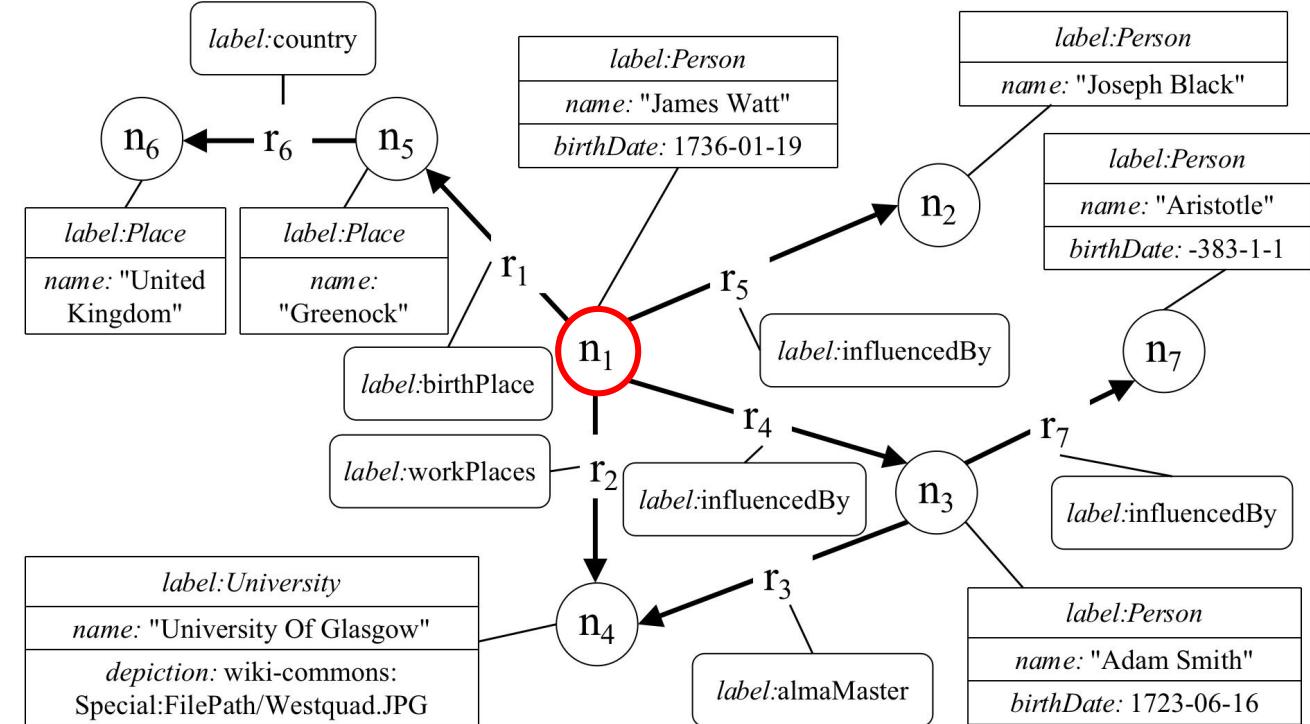
```
g.V().has("name","\"James Watt\"").  
out("influencedBy").  
values("birthDate").order().by().limit(1)
```



# Gremlin图遍历语言

## 简单遍历示例

```
g.V().has("name","\"James Watt\"").  
out("influencedBy").  
values("birthDate").order().by().limit(1)
```



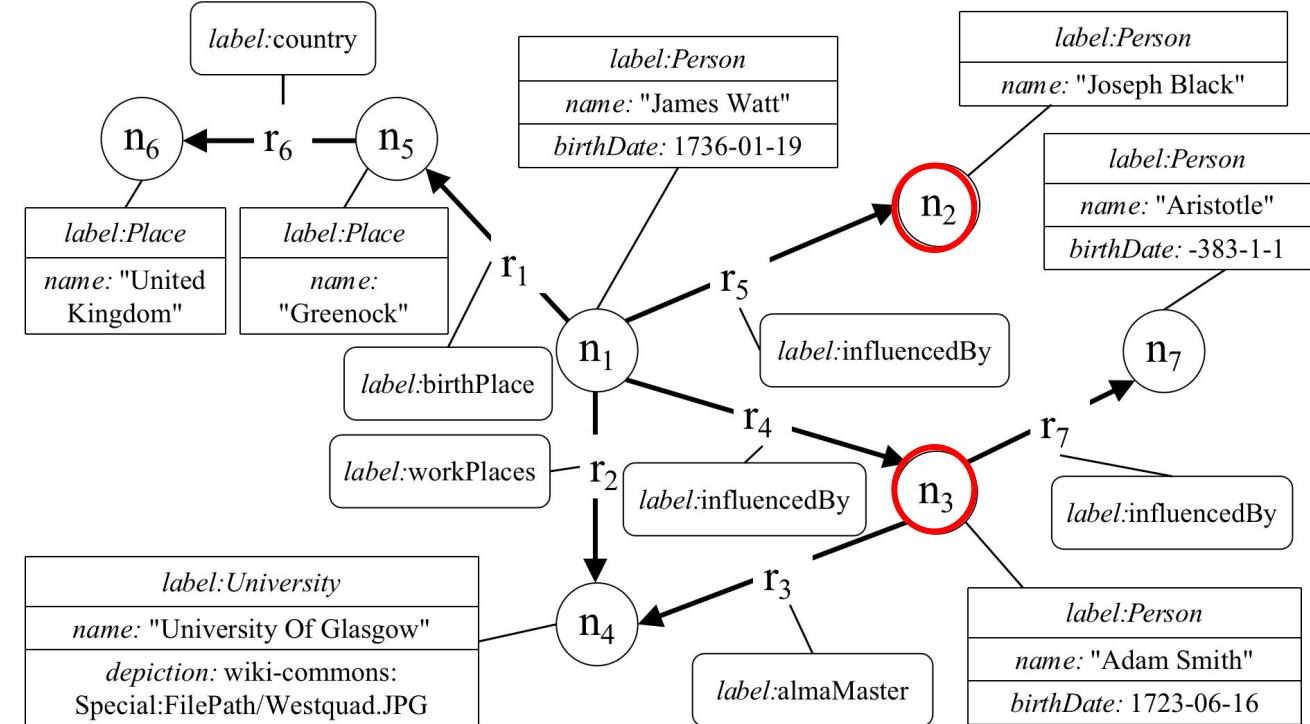
# Gremlin图遍历语言

## 简单遍历示例

```
g.V().has("name","\"James Watt\"").
```

```
out("influencedBy").
```

```
values("birthDate").order().by().limit(1)
```



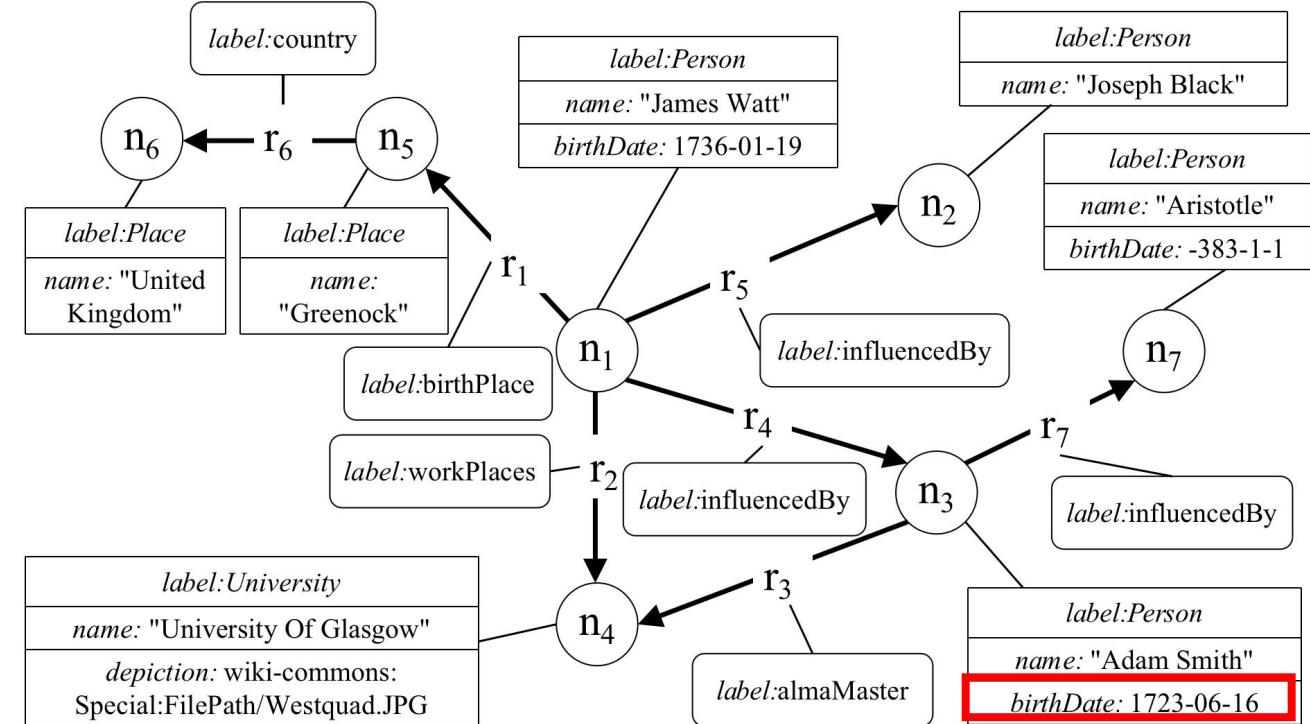
# Gremlin图遍历语言

## 简单遍历示例

```
g.V().has("name","\"James Watt\"").  
out("influencedBy").  
values("birthDate").order().by().limit(1)
```

结果输出： 123 —

06 — 16



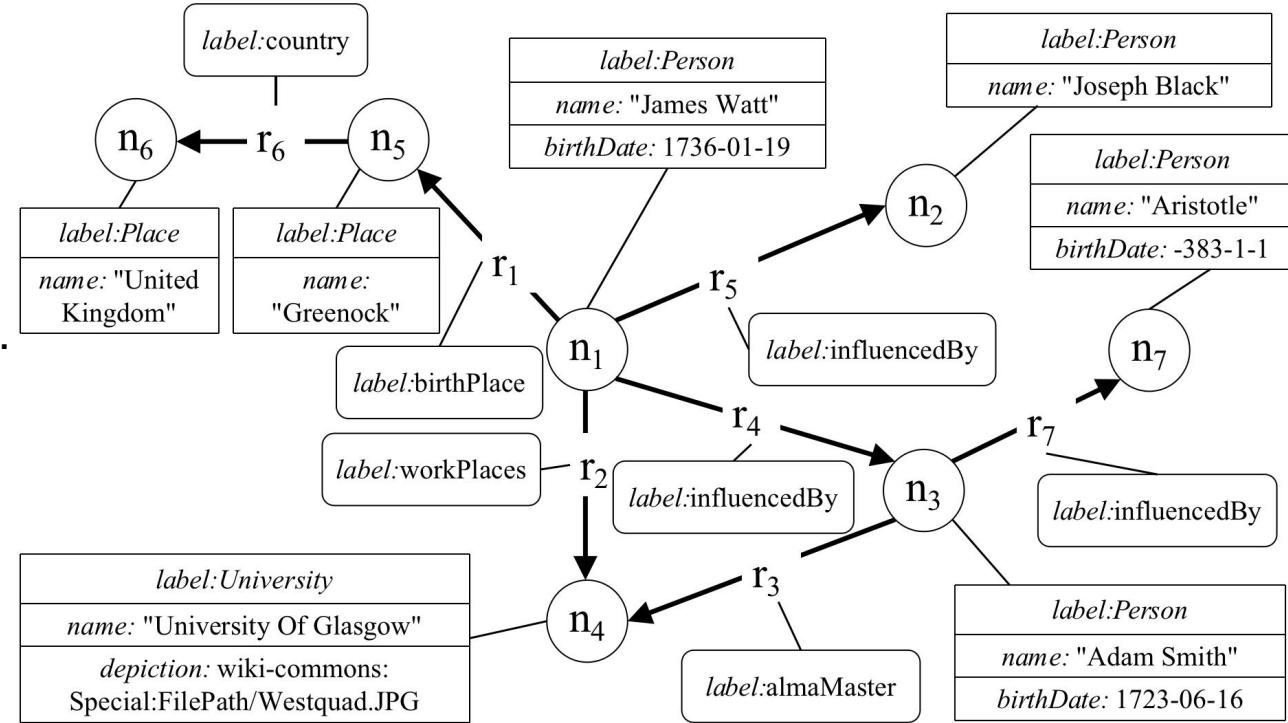
## 分支遍历

- 用来根据遍历器状态进行条件判断来确定遍历器接下来执行哪个遍历
- 步骤choose()是一个分支步骤用来提供类似if/else if/.../else 的编程结构，其通过option选项确定所形成的分支函数
- option是所谓的**步骤调制器 (step modulator)**，这种步骤调制器对语言的功能没有影响，只是用来粘结之前和之后的步骤，能更方便程序员使用

# Gremlin图遍历语言

## 分支遍历示例

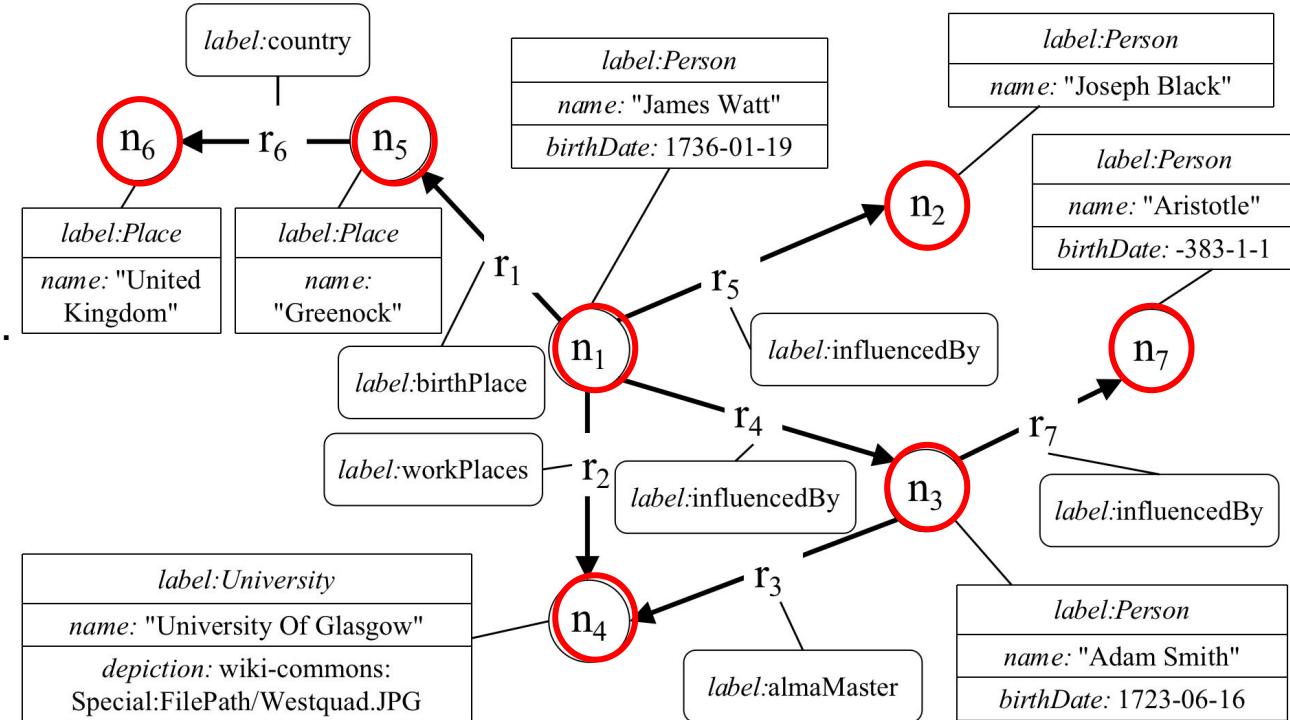
```
g.V().choose(label()).  
  option("Person", __.out("workPlaces").count()).  
  option("University", __.in("workPlaces").count()).  
  option(null, label())
```



# Gremlin图遍历语言

## 分支遍历示例

```
g.V().choose(label()).  
option("Person", __.out("workPlaces").count()).  
option("University", __.in("workPlaces").count()).  
option(None, label())
```



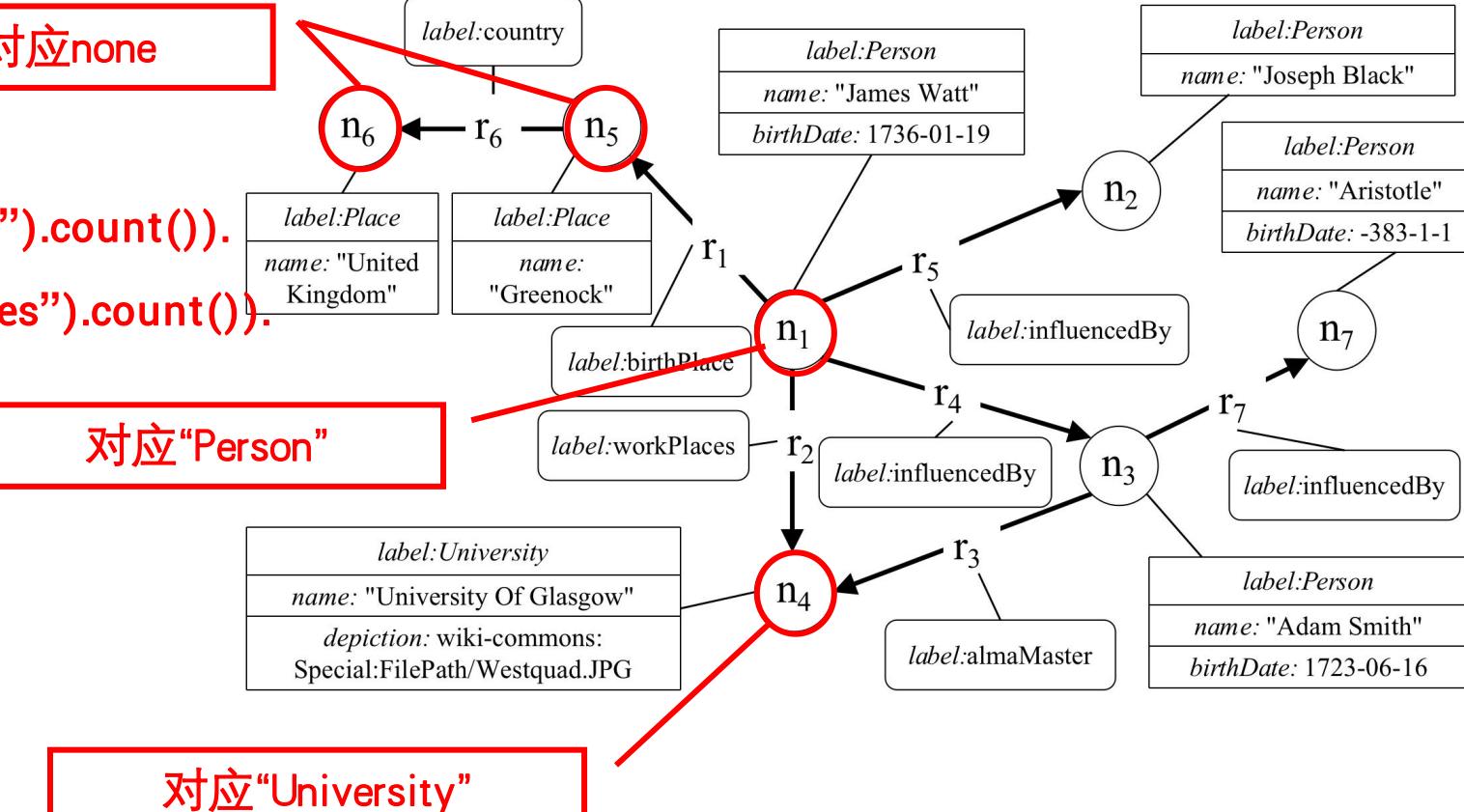
# Gremlin图遍历语言

## 分支遍历示例

```
g.V().choose(label()).  
option("Person", __.out("workPlaces").count()).  
option("University", __.in("workPlaces").count()).  
option(none, label())
```

结果输出：

1  
Place  
Place  
1



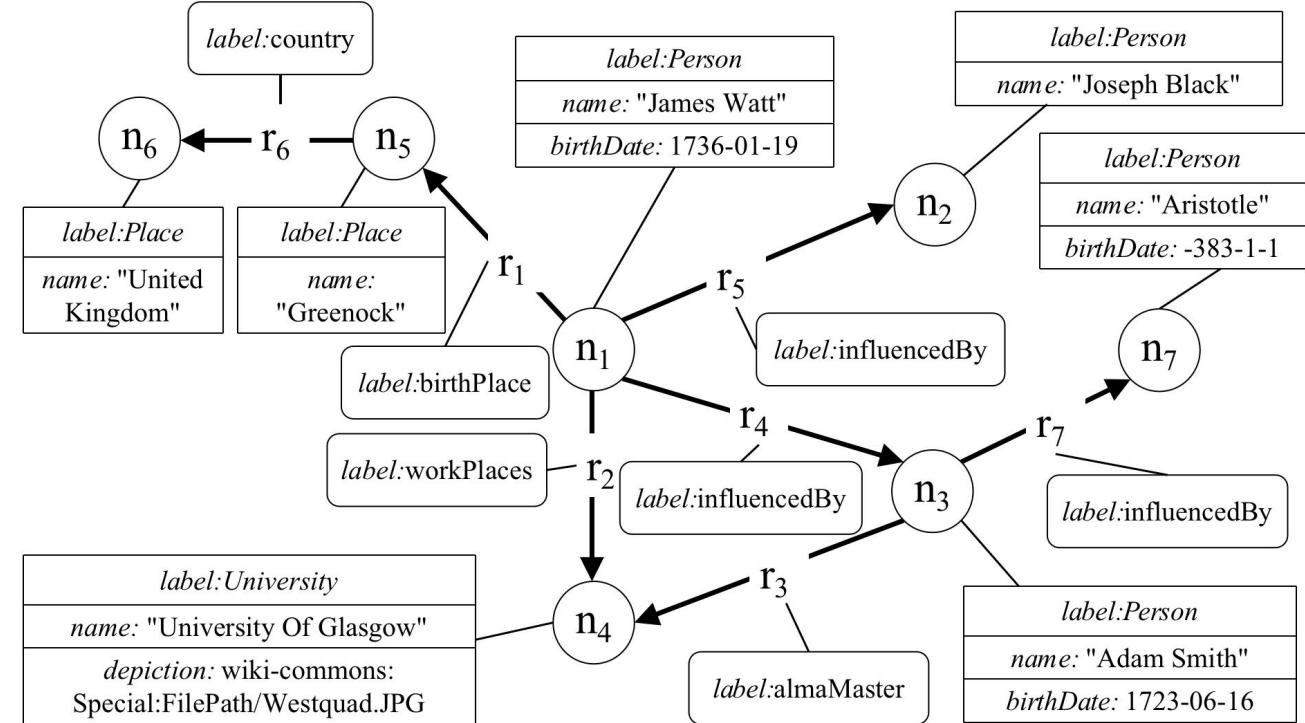
## 循环遍历

- Gremlin 图遍历语言可以利用repeat步骤定义带循环的遍历操作，并利用步骤调制器times() 来指定循环次数
- 为了支持循环，遍历器里的遍历需要可以让遍历器回到之前执行过的步骤

# Gremlin图遍历语言

## 循环遍历示例

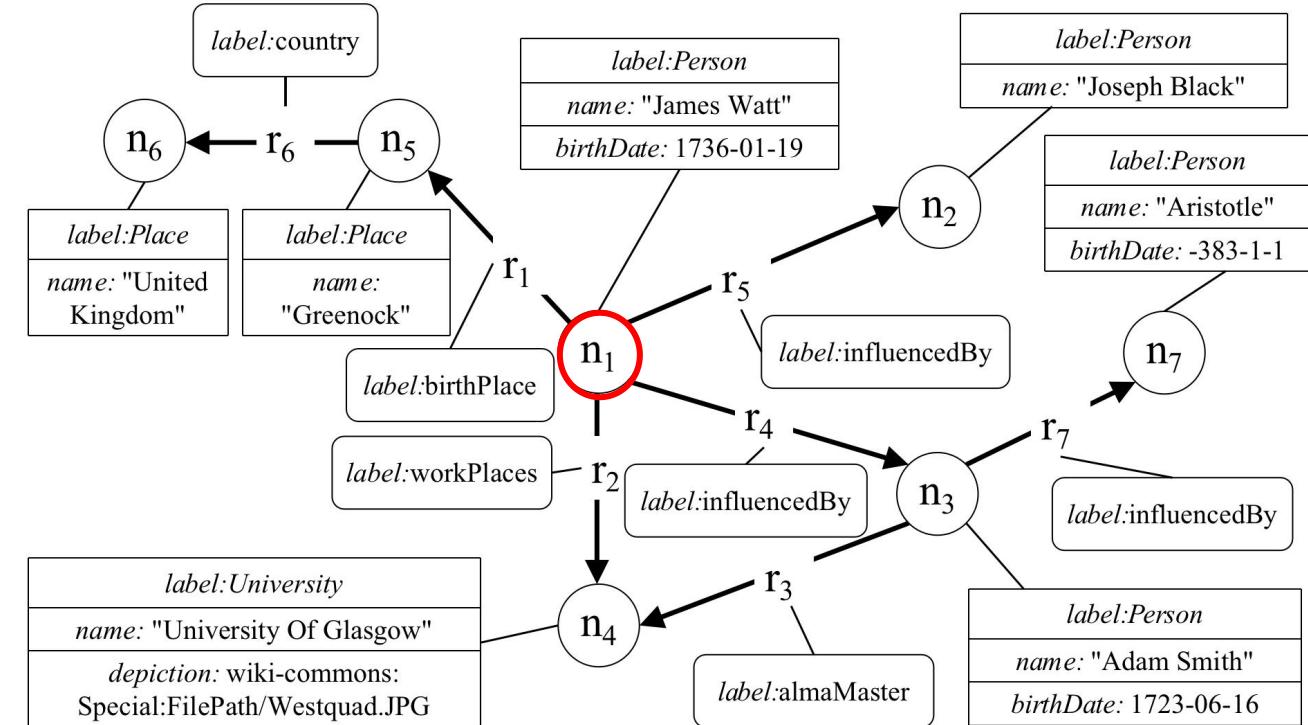
```
g.V().has("name", "JamesWatt").  
repeat(out()).times(2).values("name")
```



# Gremlin图遍历语言

## 循环遍历示例

```
g.V().has("name", "JamesWatt").  
repeat(out()).times(2).values("name")
```

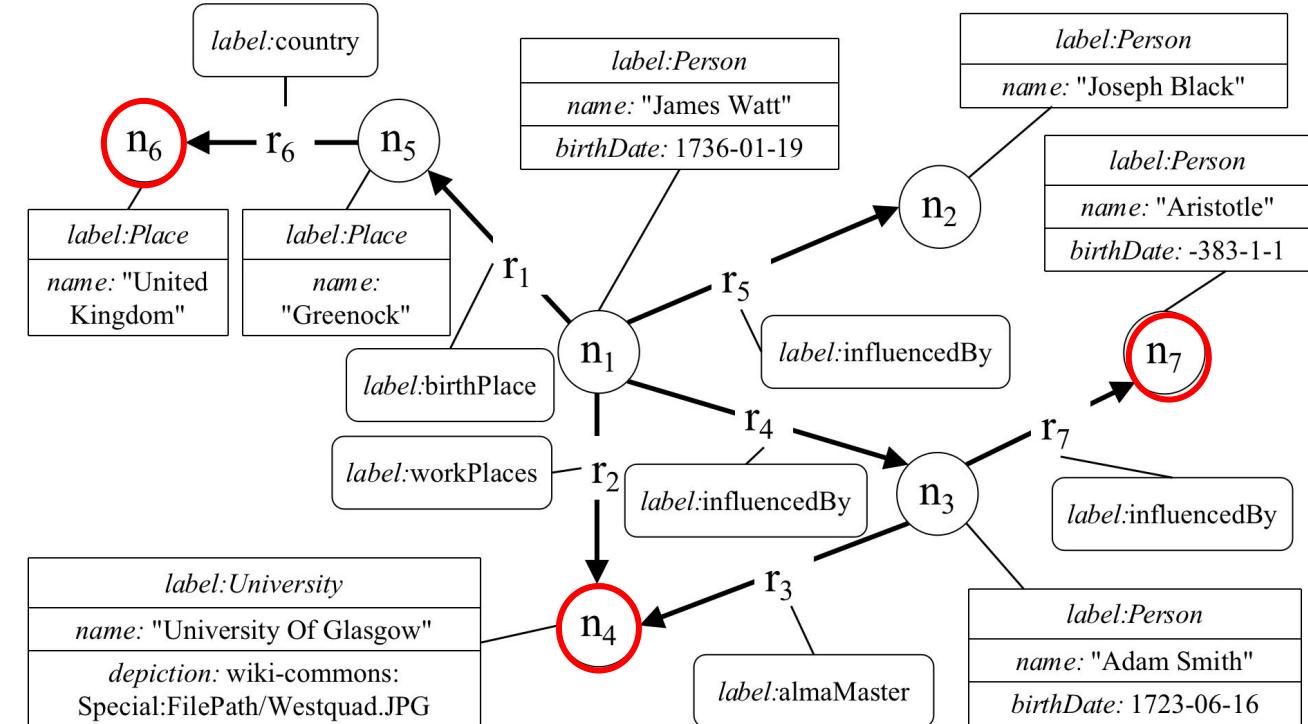


# Gremlin图遍历语言

## 循环遍历示例

```
g.V().has("name", "JamesWatt").  
repeat(out()).times(2).values("name")
```

结果输出：  
“Aristotle”  
“United Kingdom”  
“University Of Glasgow”



# Gremlin图遍历语言

## 循环遍历

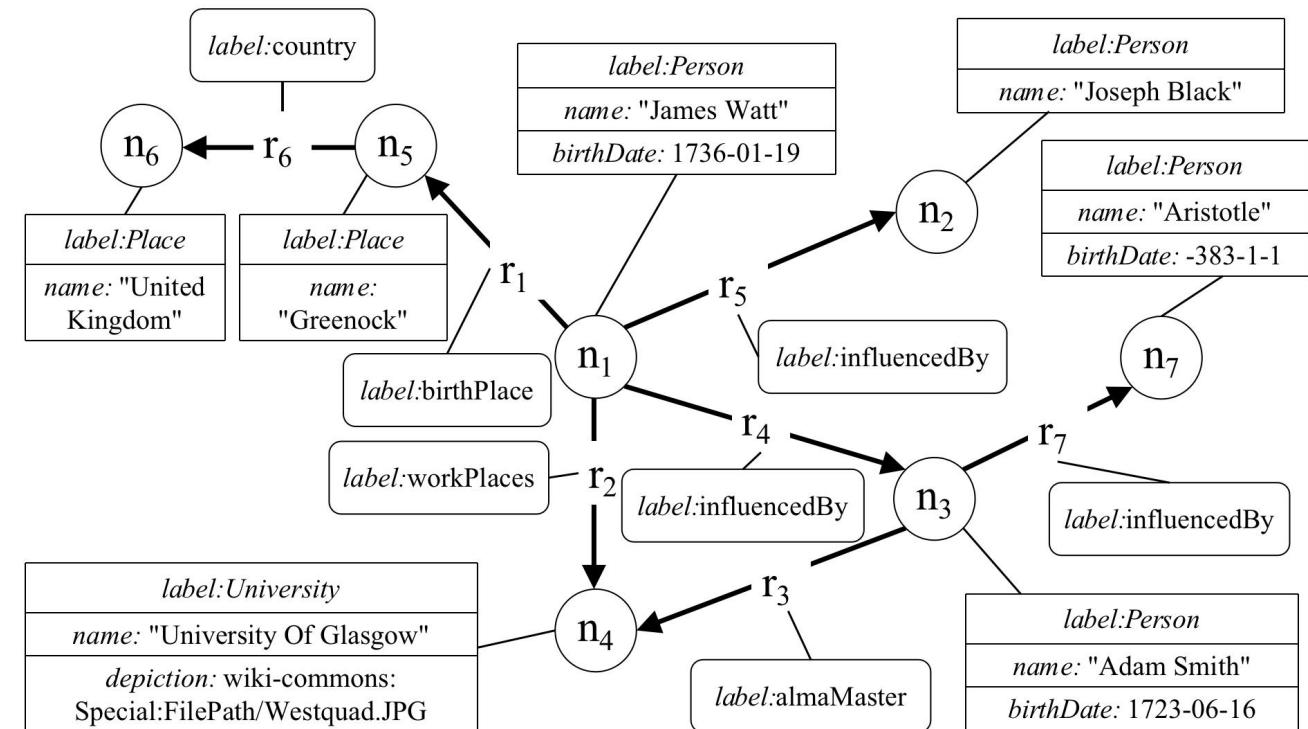
- 如果我们需要不仅仅返回“James Watt”2 步可达的点的姓名还想返回“James Watt”2 步内（含2 步）可达的所有点的姓名，那么可以利用步骤调制器emit()

```
g.V().has("name", "\\"JamesWatt\\").
```

```
repeat(out()).emit().times(2).values("name")
```

结果输出：

“Joseph Black”  
“Adam Smith”  
“Greenock”  
“University Of Glasgow”  
“University Of Glasgow”  
“Aristotle”  
“United Kingdom”



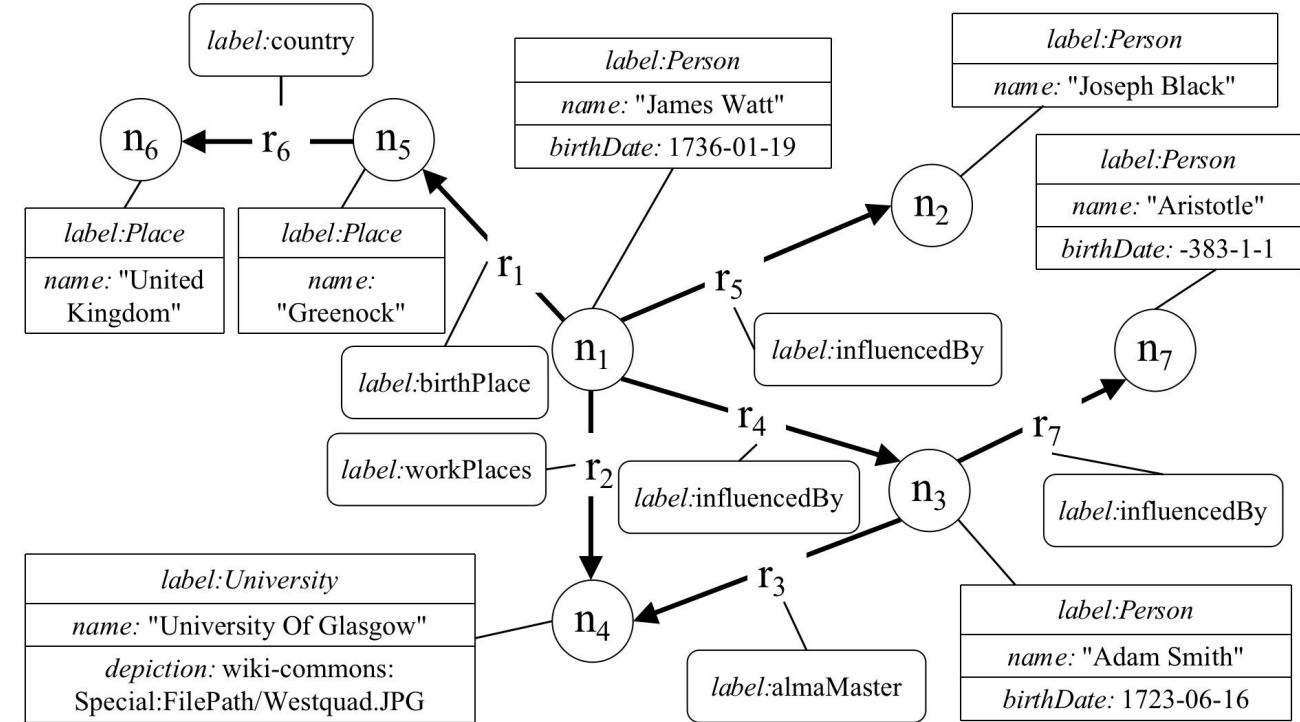
## 路径遍历

- 遍历器中还存储了**标签路径**
- 标签路径可以用来Gremlin图遍历语言返回路径。当一个遍历器走到图上的一个新的位置时，这个图上位置以及这个遍历器当前遍历的标签集合会被加入到这个遍历器的标签路径中
- 这个标签路径可以通过path()步骤来得到。

# Gremlin图遍历语言

## 路径遍历示例

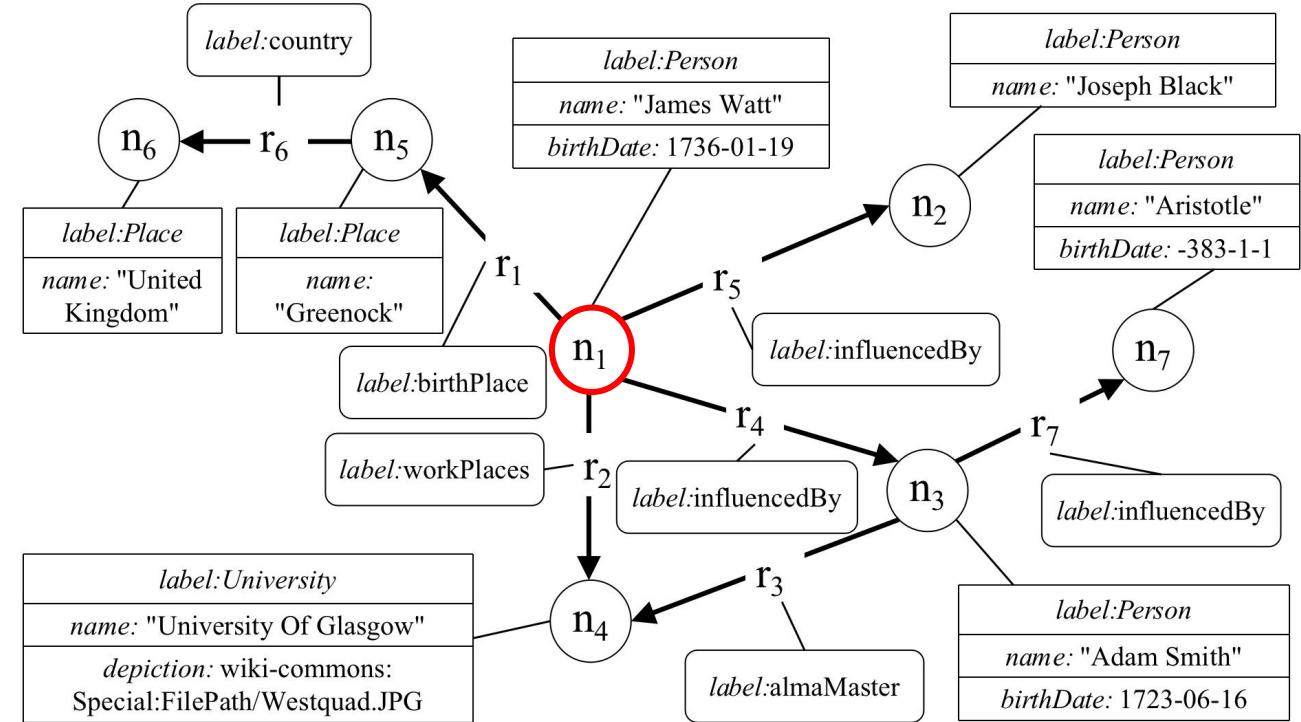
```
g.V().has("name", "JamesWatt").  
out().out().path().by(label)
```



# Gremlin图遍历语言

## 路径遍历示例

```
g.V().has("name", "JamesWatt").  
out().out().path().by(label)
```



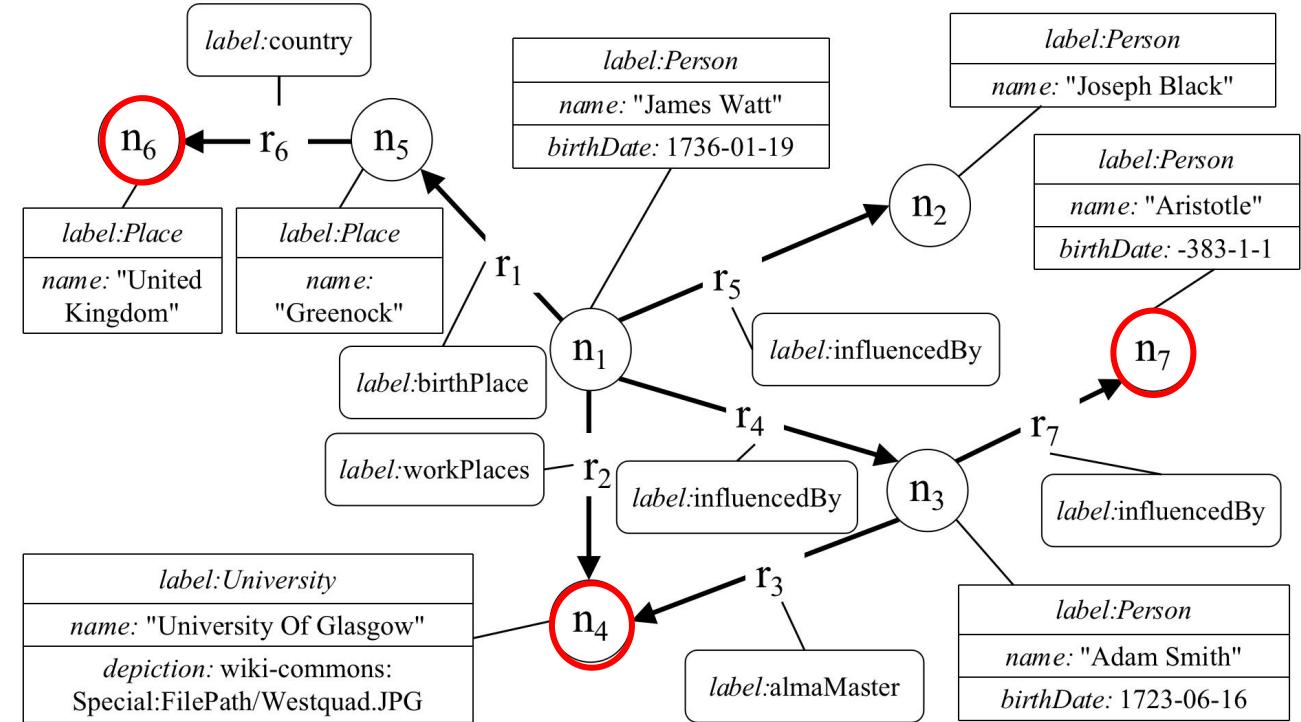
# Gremlin图遍历语言

## 路径遍历示例

```
g.V().has("name", "JamesWatt").  
out().out().path().by(label)
```

结果输出：

[Person, Person, University]  
[Person, Person, Person]  
[Person, Place, Place]



# Gremlin图遍历语言

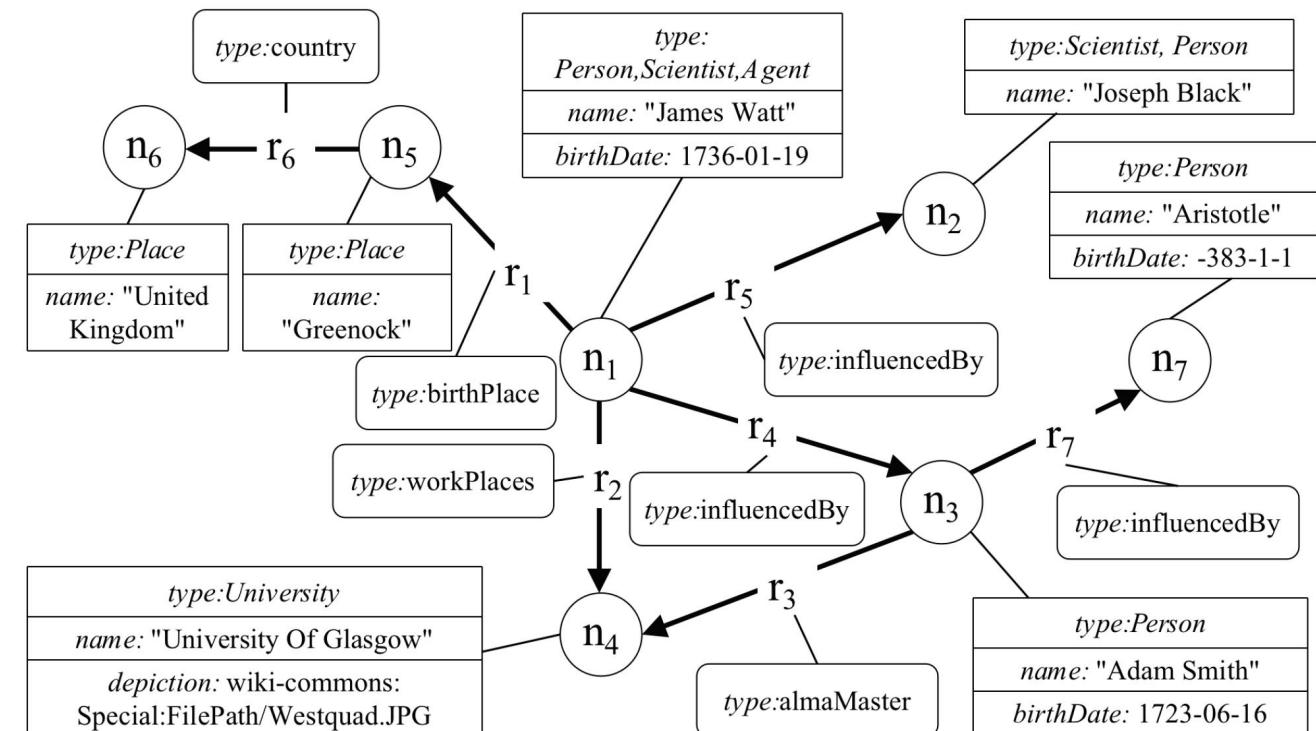
## 路径遍历

- 可以用until() 来限定repeat() 步骤的步骤调制器，整个repeat 过程中形成的路径通过simplePath 这个过滤条件来限定为简单路径，于是以下查询用来查询终点为“Person”点的最短路径

```
g.V().has("name", "\"James Watt\"").  
repeat(out()).until(label().is("Person")).  
simplePath().path().values(name).limit(1)
```

结果输出：

**["James Watt", "Joseph Black"]**



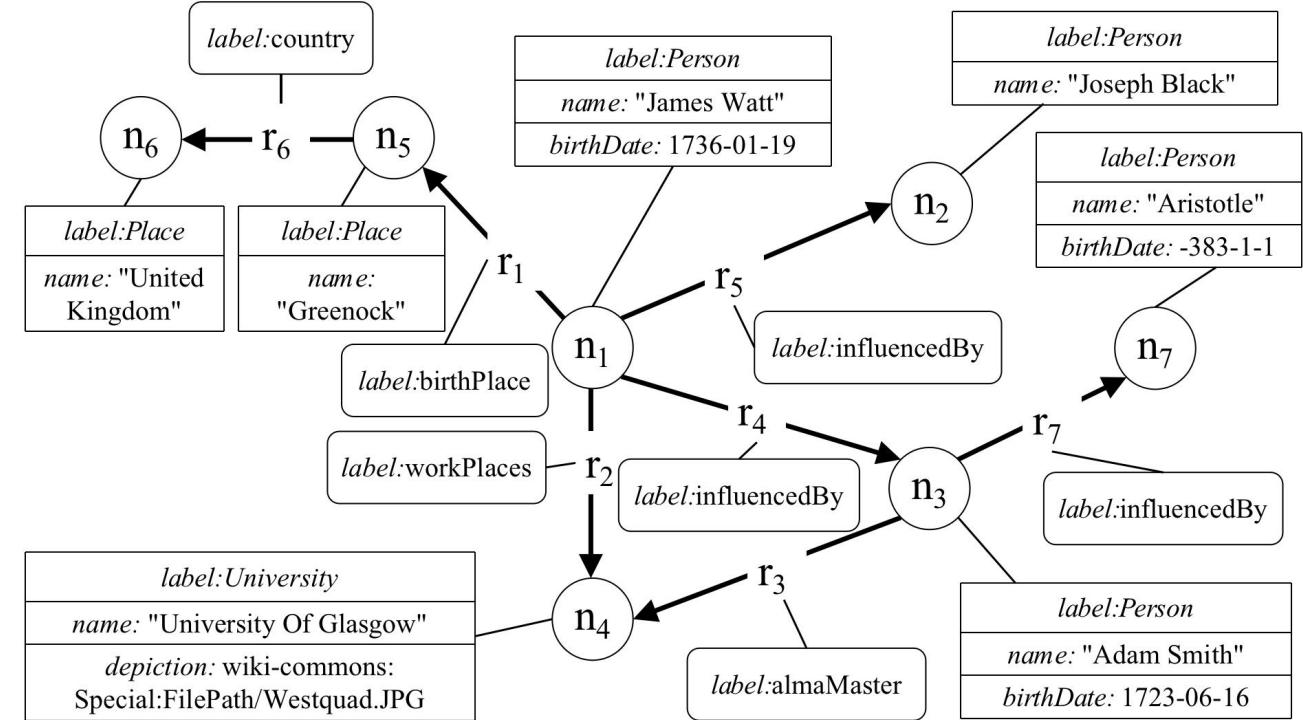
## 映射遍历

- 在实际应用中，有时候会需要遍历器回到遍历器路径历史上的某些位置，可以用select() 步骤

# Gremlin图遍历语言

## 映射遍历示例

```
g.V().has("name", "JamesWatt").  
as("a").out("influencedBy").as("b").  
select("a", "b").  
by(_.in("influencedBy").count()).  
by(_.out("influencedBy").count())
```



# Gremlin图遍历语言

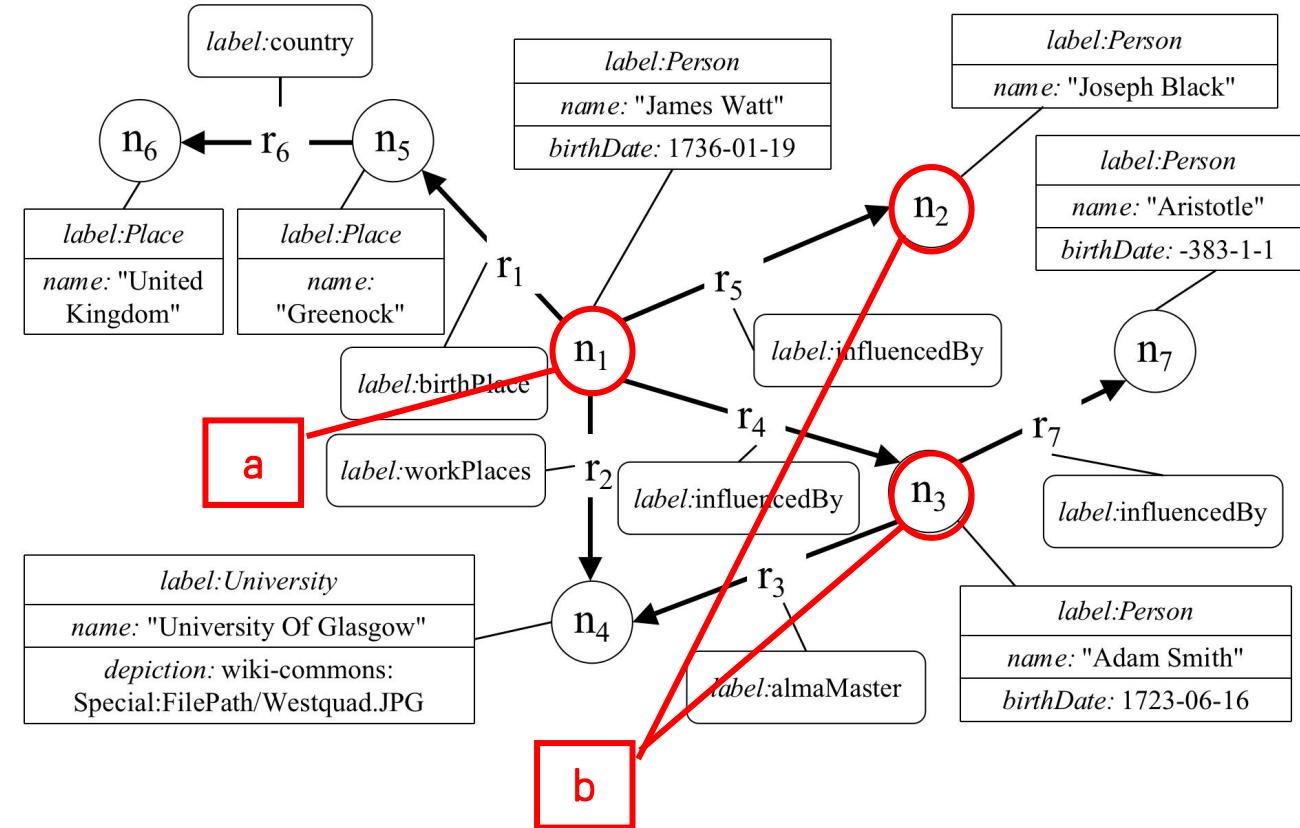
## 映射遍历示例

```
g.V().has("name", "JamesWatt").  
as("a").out("influencedBy").as("b").  
select("a", "b").  
by(__.in("influencedBy").count()).  
by(__.out("influencedBy").count())
```

结果输出：

[a:0, b:0]

[a:0, b:1]



## 变异遍历

- Gremlin 图遍历语言也支持变异遍历来对图数据进行修改，包括增删图上对图上的点、边和属性，下面列出来了一部分变异遍历

$addOutE : V^* \rightarrow E^*$

$addInE : V^* \rightarrow E^*$

$addV : U^* \rightarrow V^*$

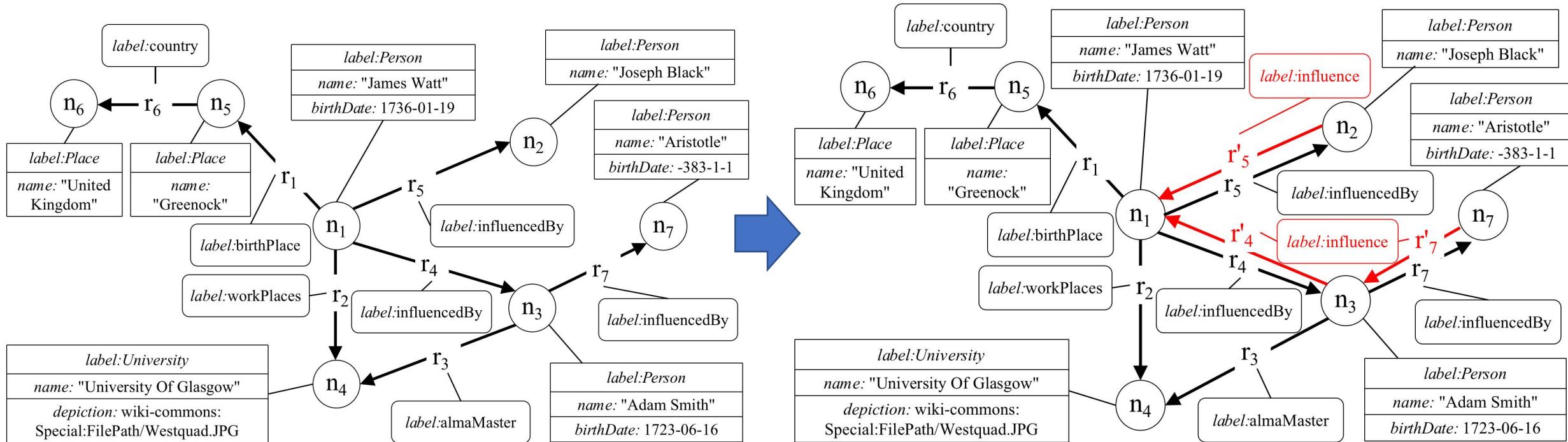
$property : (V \cup E)^* \rightarrow (V \cup E)^*$

$drop : [(V \cup E)^*] \rightarrow 0$

# Gremlin图遍历语言

## 变异遍历示例

```
g.V().as("a").out("influencedBy").addE("influence").to("a")
```



## 声明式遍历

- Gremlin图遍历语言也是支持用类似于SPARQL的声明式查询进行图模式匹配
- Gremlin图遍历语言使用match() 步骤来实现用一个命令式遍历支持声明式图查询匹配
- match() 步骤的参数是一个图模式集合，这些图模式可以通过not()、and() 以及or() 相互组合
- 当一个遍历器进入match() 步骤之后，这个遍历器会沿着这些图模式依次执行，直到最后整个图模式的匹配找到。匹配找到之后会连同其中的变量名被放到结果遍历器的标签路径中。

# Gremlin图遍历语言

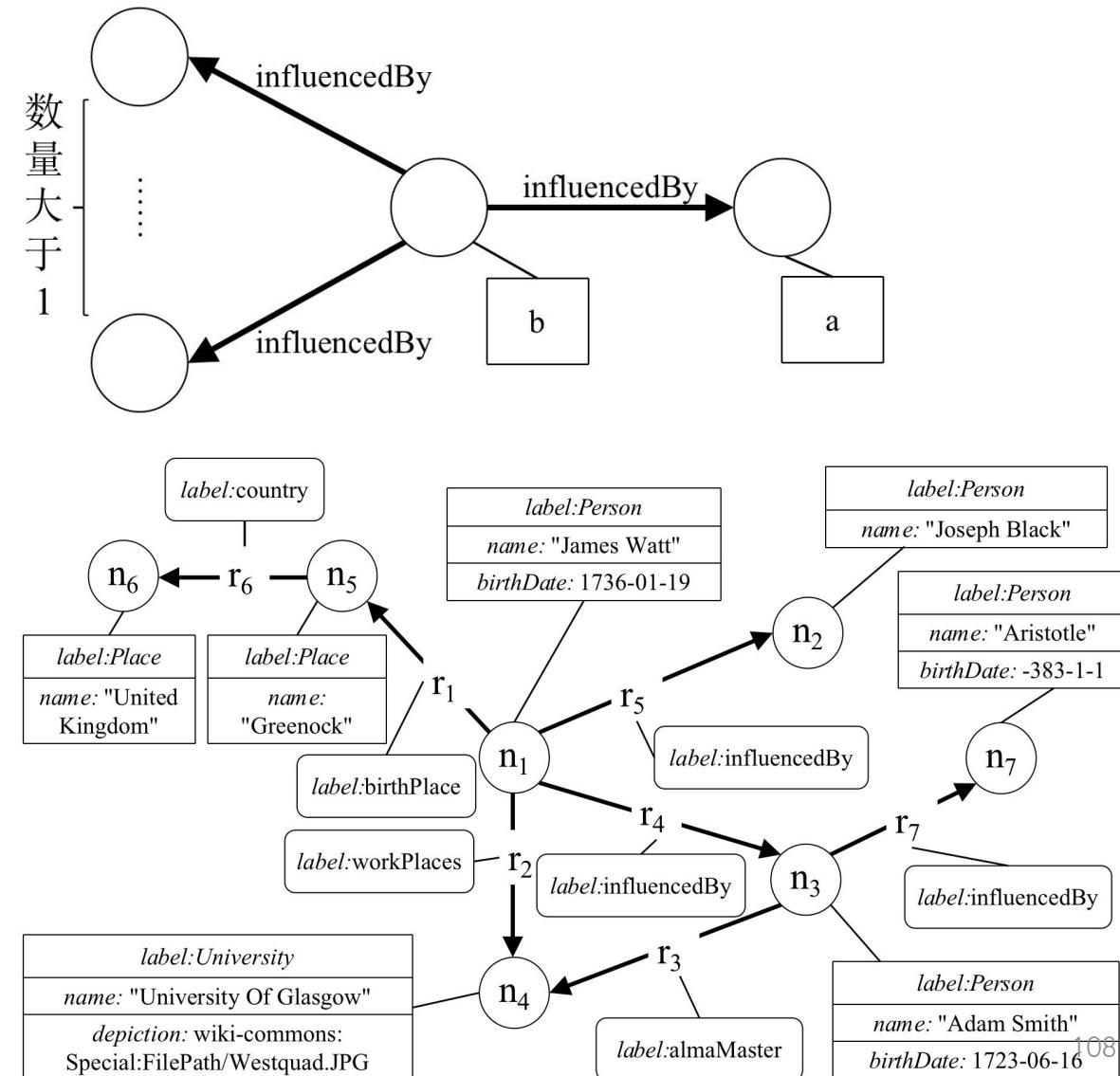
## 声明式遍历示例

```
g.V().match(  
    __.as("a").in("influencedBy").as("b"),  
    __.as("b").out("influencedBy").count().is(gt(1))  
).select("a").by("name").dedup()
```

## 结果输出：

## “Joseph Black”

## “Adam Smith”



## 定义域限定式遍历

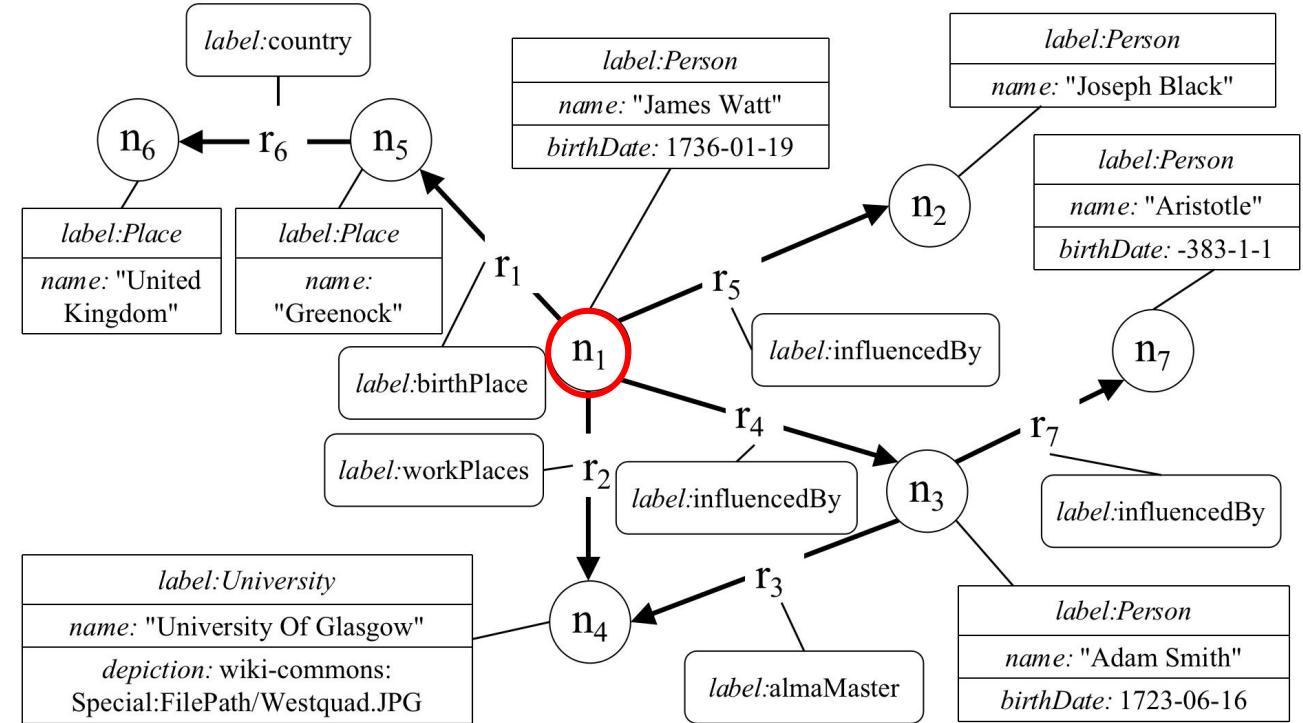
- Gremlin 图遍历机执行大概30 个定义域限定式遍历
- 这些步骤用来限定遍历器需要满足的限制条件。这些步骤也就是之前也用到过的步骤，如has()、out()

# Gremlin图遍历语言

## 定义域限定式遍历示例

```
g.V().hasLabel(Person).  
has(name, "JamesWatt")
```

结果输出：  
[n<sub>1</sub>]



# Gremlin图遍历机

---

## 图遍历机简介

- 在理论上，Gremlin 图遍历语言的目的是让用户定义Gremlin 图遍历机
- Gremlin 图遍历机是由三部分组成：图G（Graph，对应于自动机中的数据）、遍历 $\Psi$ （Traversal，对应于自动机中的指令）和遍历器T（Traverser，对应于自动机中的读/写头）

## 图遍历机——图

- 所谓图，如前文所述，在Gremlin 中所采用的图是属性图，被定义为三元组( $V$ ,  $E$ ,  $\lambda$ )

## 图遍历机——遍历

- 所谓**遍历**，在Gremlin 图遍历机中是一个函数树，其中每一个函数也被称为步骤
- 每个步骤 $f: A^* \rightarrow B^*$ 是一个从遍历器集合 $A^*$ 到遍历器集合 $B^*$ 的函数， $A^*$ （或 $B^*$ ）表示位于全集 $U$  中类型是 $A$ （或 $B$ ）的对象上的遍历器的集合
- 步骤的组合也是步骤。步骤组合方式有两种：**链式**和**递归式**。所谓链式，就是给定三个步骤 $f$ 、 $g$ 、 $h$ ，那么 $f \circ g \circ h$ 也是步骤；所谓递归式，就是给定三个步骤 $f$ 、 $g$ 、 $h$ ，那么 $f(g \circ h)$ 也是步骤。

## 图遍历机——遍历

- Gremlin 中步骤可以分为以下五类：
  1. map 型步骤，这种步骤是 $A^* \rightarrow B^*$ ，且定义域和值域大小一样。这种步骤将每个位于类型是A的对象上的遍历器映射到一个位于类型是B的对象上的遍历器；
  2. flatMap 型步骤，这种步骤是 $A^* \rightarrow B^*$ ，且定义域和值域不限定。这种步骤将每个位于类型是A的对象上的遍历器映射到零个、一个或者多个位于类型是B的对象上的遍历器；

## 图遍历机——遍历

3. filter 型步骤，这种步骤是 $A^* \rightarrow A^*$ ，且值域是定义域的子集。这种步骤将位于类型是 $A$ 的对象上的遍历器按照过滤条件过滤掉一些；
4. sideEffect 型步骤，这种步骤是 $A^* \rightarrow_x A^*$ ，且定义域和值域一样。这类步骤对位于类型是 $A$ 的对象上的遍历器通过定义在图上数据结构 $x$  执行一些函数；
5. branch 型步骤，这种步骤是 $A^* \rightarrow^b B^*$ 。其中， $b: T \rightarrow \mathcal{P}(\Psi)$ 是一个将遍历器映射到不同步骤的函数， $\mathcal{P}(\Psi)$ 表示 $\Psi$ 的幂集。这类步骤表示根据不同遍历器选择不同的函数进行执行

## 图遍历机——遍历器

- 所谓**遍历器**，在Gremlin 图遍历机中是一个六元组集合

$$T \subseteq (U \times \Psi \times (\mathbb{P}(\Sigma^*) \times U)^* \times \mathbb{N}^+ \times U \times \mathbb{N}^+)$$

- 第一项为遍历器在图上的位置，记为 $\mu(t)$ 。 $\mu : T \rightarrow U$  表示将遍历器映射到一个图上的对象；
- 第二项为遍历器目前所处在的遍历，记为 $\psi(t)$ 。 $\psi : T \rightarrow \Psi$  表示将遍历器映射到一个遍历

## 图遍历机——遍历器

- 所谓**遍历器**，在Gremlin 图遍历机中是一个六元组集合

$$T \subseteq (U \times \Psi \times (\mathbb{P}(\Sigma^*) \times U)^* \times \mathbb{N}^+ \times U \times \mathbb{N}^+)$$

- 第三项为标签路径，记为 $\Delta(t)$ 。 $\Delta: T \rightarrow (\mathbb{P}(\Sigma^*) \times U)^*$ 将遍历器映射到一个二元对序列，表示遍历到遍历器目前状态所经过的图上元素以及其标签。比如标签路径 $((a, x), (b, c), (\Phi, z))$ ，表示走到遍历器目前状态经过了 $x \rightsquigarrow y \rightsquigarrow z$ 这条图上路径，且遍历器在经过 $x$ 、 $y$  和 $z$  的时候执行遍历中的步骤后结果分别为 $(a)$ 、 $(b, c)$  和 $\Phi$ ；

# Gremlin图遍历机

---

## 图遍历机——遍历器

- 所谓**遍历器**，在Gremlin 图遍历机中是一个六元组集合

$$T \subseteq (U \times \Psi \times (\mathbb{P}(\Sigma^*) \times U)^* \times \mathbb{N}^+ \times U \times \mathbb{N}^+)$$

- 第四项为遍历器中进行批处理的遍历器数量 (bulk) , 记为 $\beta(t)$ 。 $\beta: T \rightarrow \mathbb{N}^+$  表示这个遍历器中是多少个遍历器的合并。 $\beta$  并不是必须的，只是一个在执行遍历的过程中将多个一样的遍历器合并执行的优化手段；
- 第五项为数据结构sack，记为 $\varsigma(t)$ 。 $\varsigma: T \rightarrow U$ 表示这个遍历器目前所对应变量；
- 第六项为循环计数，记为 $\iota(t)$ 。 $\iota: T \rightarrow \mathbb{N}^+$ 记录这个遍历器在带循环的遍历中走过了多少循环。

# Gremlin图遍历机

---

## 图遍历机复杂度

- 理论上Gremlin 图遍历机同构于一个单头图灵机。

# THANK YOU

---