第5章 Neo4j图数据库

张元鸣

计算机学院

本章内容

5.1 图数据库概述

5.2 Neo4j的基本元素

5.3 Neo4j的基本操作

5.4 Neo4j的函数

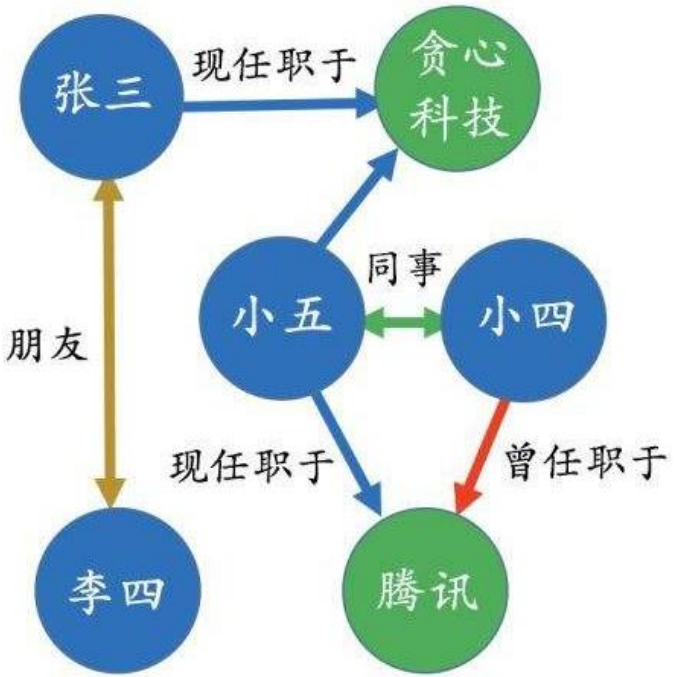
5.5 Neo4j的模式操作

5.6 Neo4j应用实例

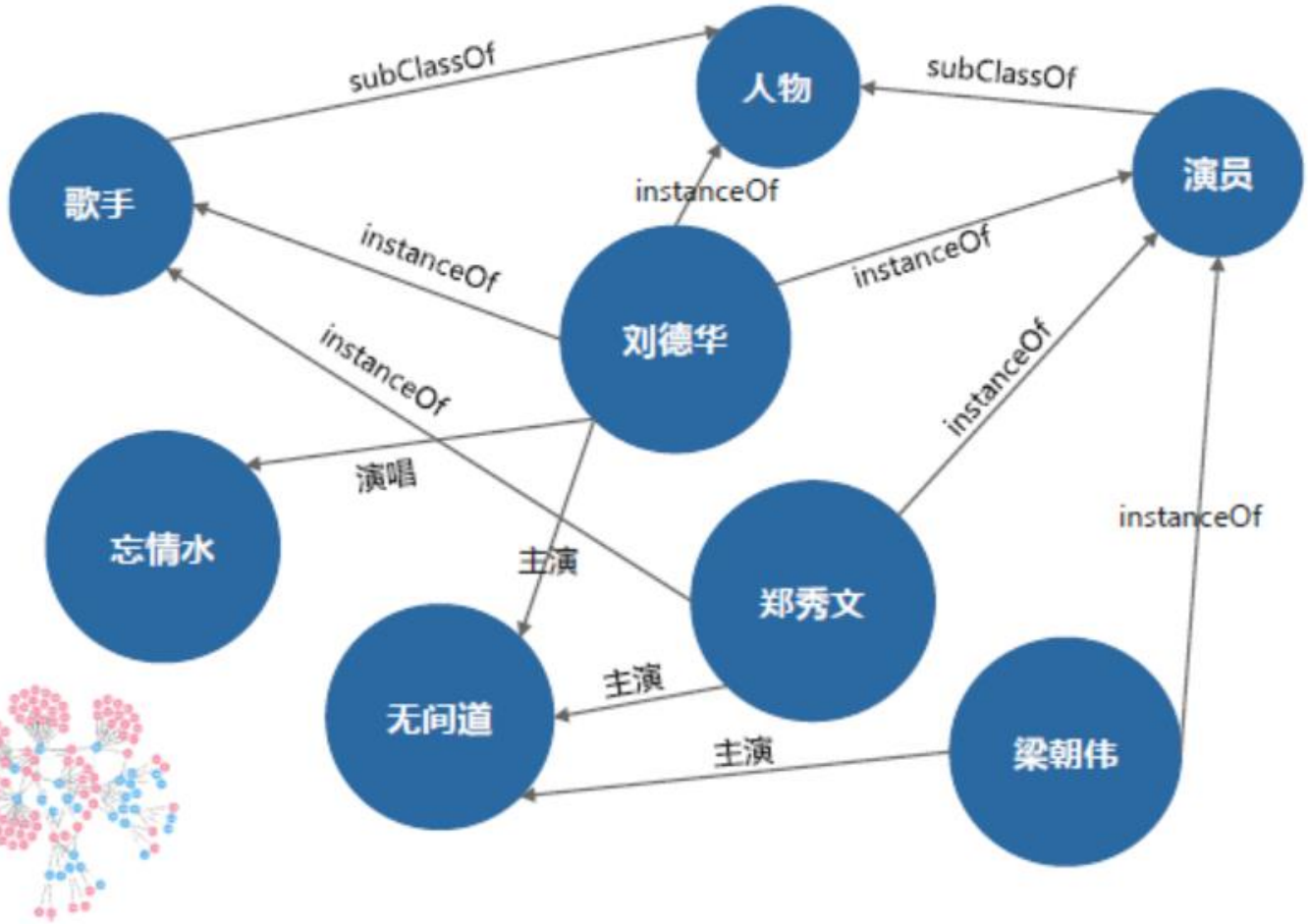
5.1 图数据库概述

图数据库是一种基于图论的新型NoSQL数据库，它的数据存储结构和数据查询方式都是以图论为基础：

- 节点：表示对象，对象可以有属性和标签；
- 边：表示对象间的关系，边也可以有属性和类型；
- 操作：写操作、读操作和通用操作。

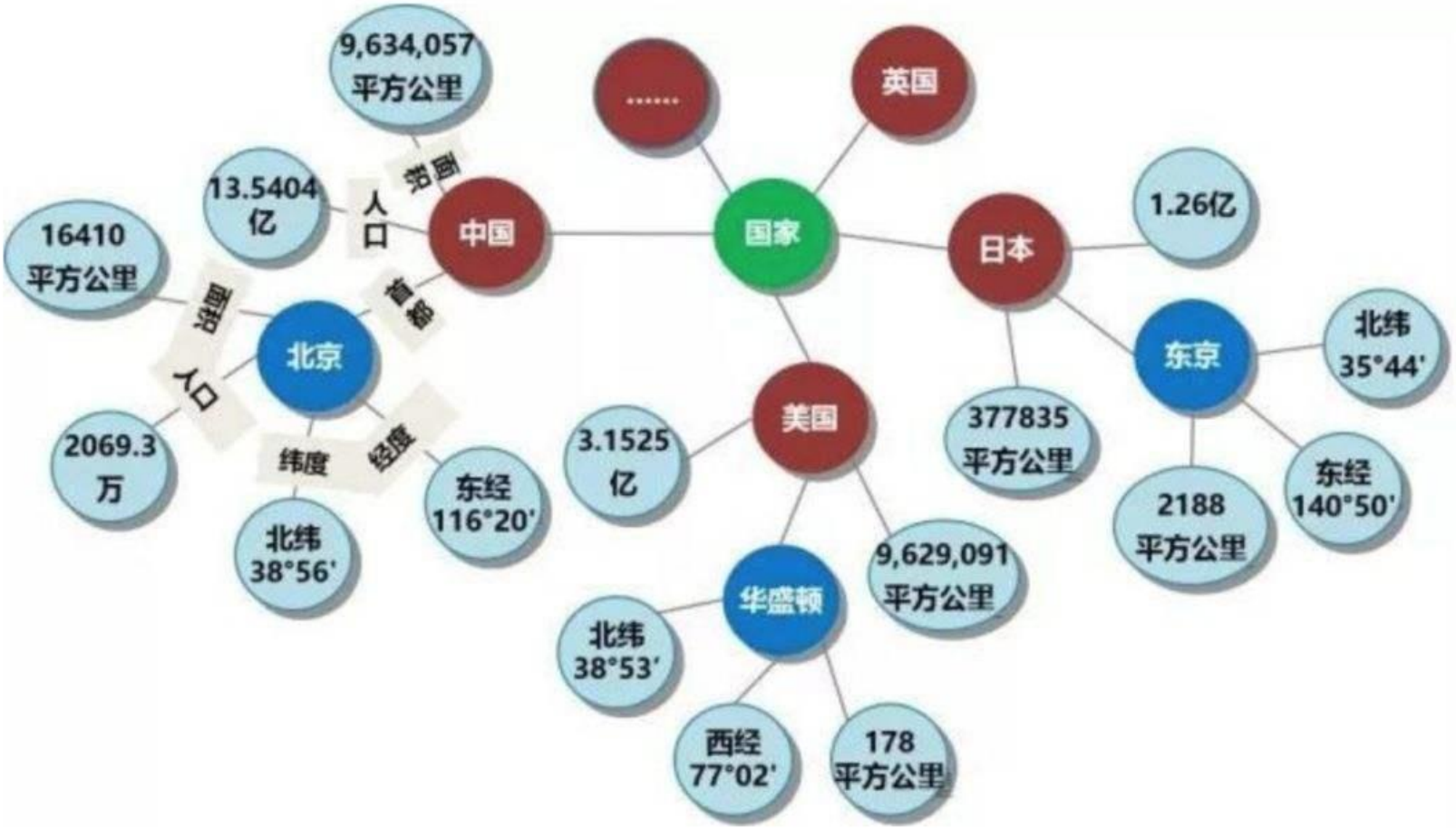


5.1 图数据库概述



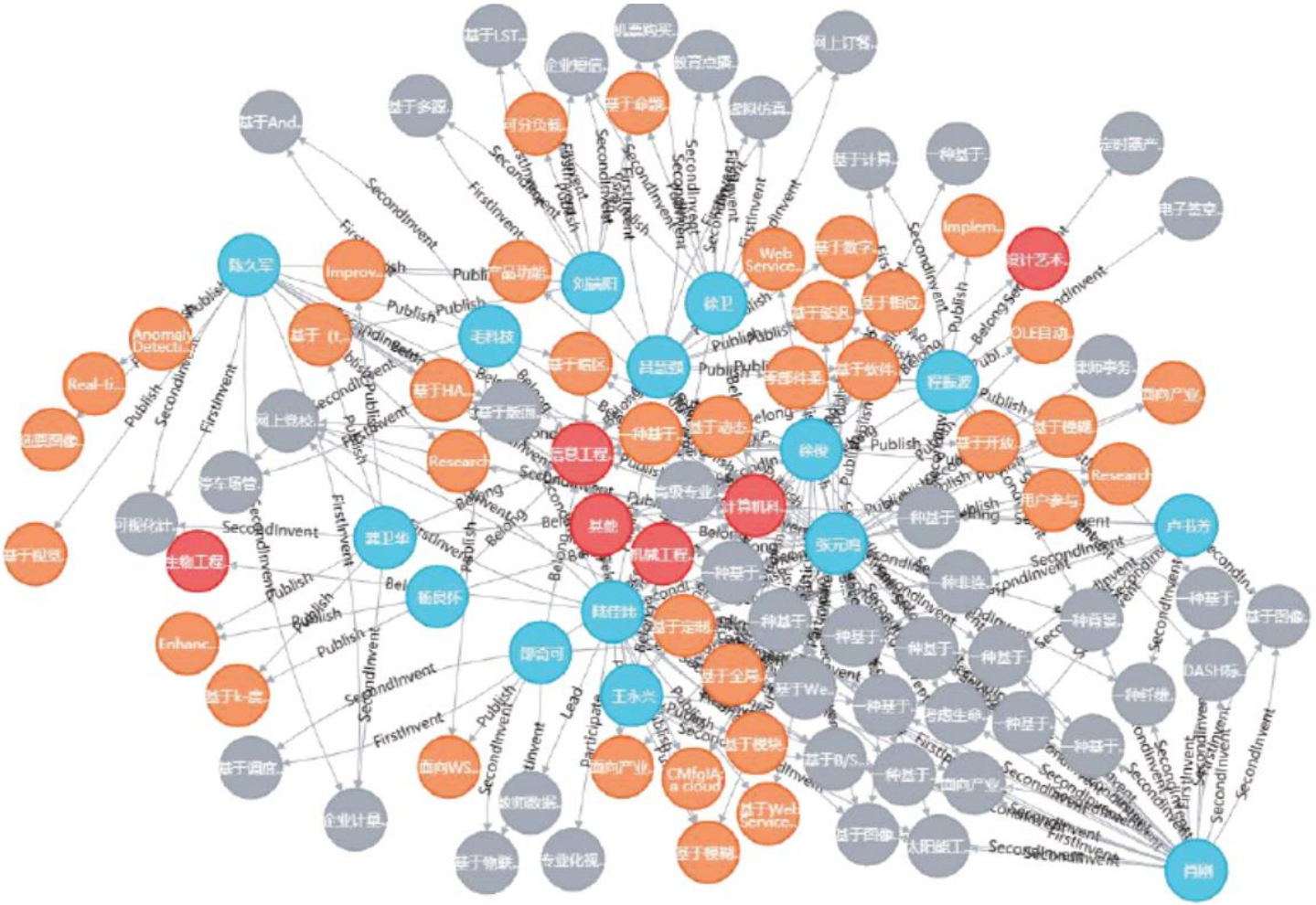
人际关系图谱

5.1 图数据库概述



国家与城市图谱

5.1 图数据库概述



学术知识图谱

软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.005841]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

知识图谱数据管理研究综述*

王 鑫[1,2], 邹 磊[3], 王朝坤[4], 彭 鹏[5], 冯志勇[1,2]

[1](天津大学 智能与计算学部, 天津 300354)

[2](天津市认知计算与应用重点实验室, 天津 300354)

[3](北京大学 计算机科学技术研究所, 北京 100080)

[4](清华大学 软件学院, 北京 100084)

[5](湖南大学 信息科学与工程学院, 长沙 湖南 410082)

通讯作者: 王鑫, E-mail: wangx@tju.edu.cn

摘 要: 知识图谱是人工智能的重要基石.各领域大规模知识图谱的构建和发布对知识图谱数据管理提出了新的挑战.以数据模型的结构和操作要素为主线,对目前的知识图谱数据管理理论、方法、技术与系统进行研究综述.首先,介绍知识图谱数据模型,包括 RDF 图模型和属性图模型,介绍 5 种知识图谱查询语言,包括 SPARQL、Cypher、

软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2018,29(10) [doi: 10.13328/j.cnki.jos.005552]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一种准确高效的领域知识图谱构建方法*

杨玉基, 许 斌, 胡家威, 仝美涵, 张 鹏, 郑 莉

(清华大学 计算机科学与技术系知识工程实验室,北京海淀 100084)

通讯作者: 杨玉基, E-mail: yangyujiyyj@gmail.com

摘要: 作为语义网的数据支撑,知识图谱在知识问答、语义搜索等领域起着至关重要的作用,一直以来也是研究领域和工程领域的一个热点问题,但是构建一个质量较高、规模较大的知识图谱往往需要花费巨大的人力和时间成本.如何平衡准确率和效率,快速地构建出一个高质量的领域知识图谱,是知识工程领域的一个重要挑战.本文对领域知识图谱构建方法做了系统研究,提出了一种准确高效的领域知识图谱构建方法——“四步法”,我们将此方法应用到中国基础教育九门学科知识图谱的构建中,在较短时间构建出了准确率

软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(10):2966−2994 [doi: 10.13328/j.cnki.jos.005551]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向知识图谱的知识推理研究进展*

官赛萍[1,2], 靳小龙[1,2], 贾岩涛[1,2], 王元卓[1,2], 程学旗[1,2]

[1](网络数据科学与技术重点实验室(中国科学院 计算技术研究所),北京 100190)

[2](中国科学院大学 计算机与控制学院,北京 100049)

通讯作者: 靳小龙, E-mail: jinxiaolong@ict.ac.cn

摘 要: 近年来,随着互联网技术和应用模式的迅猛发展,引发了互联网数据规模的爆炸式增长,其中包含大量有价值的知识.如何组织和表达这些知识,并对其进行深入计算和分析备受关注.知识图谱作为丰富直观的知识表达方式应运而生.面向知识图谱的知识推理是知识图谱的研究热点之一,已在垂直搜索、智能问答等应用领域发挥了重要作用.面向知识图谱的知识推理旨在根据已有的知识推理出新的知识或识别错误的知识.不同于传统的知识推理,由于知识图谱中知识表达形式的简洁直观、灵活丰富,面向知识图谱的知识推理方法也更加多样化.将从知识推理的基本概念出发,介绍近年来面向知识图谱知识推理方法的最新研究进展.具体地,根据推理类型划分,将面向知识

第 14 卷第 2 期 智 能 系 统 学 报 Vol.14 No.2
2019 年 3 月 CAAI Transactions on Intelligent Systems Mar. 2019

DOI: 10.11992/tis.201805001
网络出版地址: http://kns.cnki.net/kcms/detail/23.1538.TP.20180702.1311.002.html

知识图谱的推荐系统综述

常亮，张伟涛，古天龙，孙文平，宾辰忠
（桂林电子科技大学 广西可信软件重点实验室，广西 桂林 541004）

摘　要：如何为用户提供个性化推荐并提高推荐的准确度和用户满意度，是当前推荐系统研究面临的主要问题。知识图谱的出现为推荐系统的改进提供了新的途径。本文研究了知识图谱近年来在推荐系统中的应用情况，从基于本体的推荐生成、基于开放链接数据的推荐生成以及基于图嵌入的推荐生成 3 个方面对研究现状进行了综述。在此基础上，提出了基于知识图谱的推荐系统总体框架，分析了其中涉及的关键技术，并对目前存在的重点和难点问题进行了讨论，指出了下一步需要开展的研究工作。

基于知识图谱关系路径的多跳智能问答模型研究

张元鸣,姬 琦,徐雪松,程振波,肖 刚
(浙江工业大学计算机科学与技术学院,浙江杭州 310023)

摘 要: 多跳问题是一类通过知识推理才能给出答案的复杂问题,往往需要相关的多项关联知识融合生成最终答案.现有基于知识图谱的多跳智能问答方法推理过程比较复杂,没有考虑关系路径蕴含的结构信息和语义信息.为此,本文提出了基于知识图谱关系路径的多跳智能问答模型,将多跳智能问答问题转换为在低维向量空间中查找知识图谱中最优关系路径的问题.该模型利用表示学习将知识图谱和用户问题同时嵌入到低维的向量空间,实现知识空间和问题空间的统一表示;然后结合主题实体向量表示和问题向量表示对候选实体进行语义评分,产生候选答案集合;以问题实体为起始节点,以候选答案实体为结束节点,从知识图谱中抽取与问题相关的关系路径集合;将关系路径进一步嵌入到低维的向量空间,生成关系路径的向量表示,在向量空间中查找与问题语义匹配度最高的关系路径,最终根据关系路径生成多跳问题的答案.在公开的数据集上对所提出的模型进行了实验,结果表明该方法与现有方法相比不仅具有良好的性能,而且具有良好的稳定性,不会随着问题跳数的增加而降低性能.

关键词: 智能问答;知识图谱;复杂多跳问题;关系路径;表示学习
基金项目: 浙江省"尖兵""领雁"研发攻关计划项目(No.2023C01022);国家自然科学基金(No.61976193)
中图分类号: TP391.1; **文献标识码:** A **文章编号:** 0372-2112(XXXX)XX-0001-08
电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20230477

基于知识图谱多集池化的健康状态智能评估方法

张元鸣，肖士易，徐雪松，程振波，肖刚+
（浙江工业大学计算机科学与技术学院，浙江杭州 310023）

摘要：为了从装备传感器监测数据和其他关联数据中提取更全面的时间域和空间域特征信息，提出了一种基于知识图谱多集池化的健康状态评估方法。构建了带时间标签的健康知识图谱，以建模装备一段时间内监测数据、部件组成数据和先验知识间的时空依赖关系。在此基础上，设计了图多集池化网络模型，该模型通过节点特征学习、第一级图池化、自注意力特征学习和第二级图池化能够生成图谱的整体向量表示，将健康状态评估转换为基于表示学习的图谱分类任务。在公开的发动机数据集上对本方法进行了实验评价，结果表明，该方法能够获得较高的评估准确度，在小样本情况下也表现出良好的优势。

关键词：健康状态评估；图神经网络；知识图谱；时空特征；图池化

Temporal Knowledge Graph Informer Network for Remaining Useful Life Prediction

Yuanming Zhang, Weiyue Zhou, Jiacheng Huang, Xiaohang Jin, *Senior Member, IEEE*, and Gang Xiao

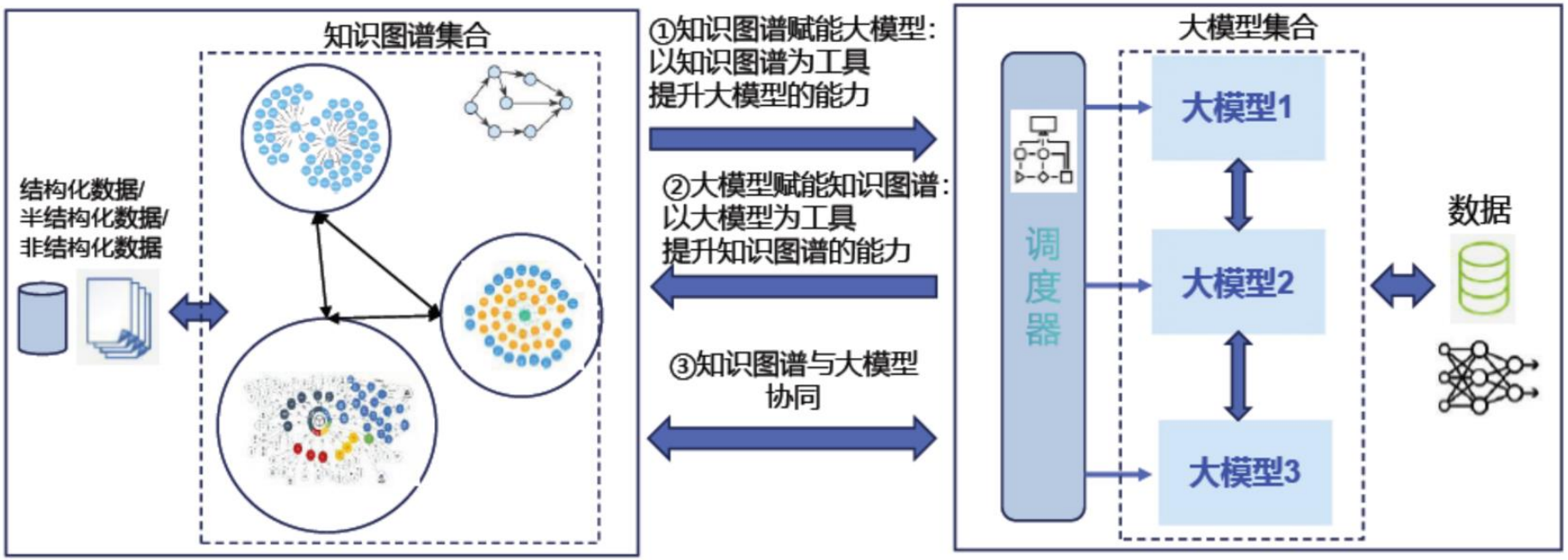
***Abstract*— Remaining useful life (RUL) prediction is of great significance to ensure the safety and reliability of equipment. Graph neural network (GNN)-based methods show great potential to improve RUL prediction performance by extracting spatiotemporal (ST) features from sensor monitoring data; however, current methods construct sensor-based homogeneous graphs without considering equipment component structure data and prior knowledge, which cannot characterize the dependency between sensors and studied equipment accurately. To solve this problem, we propose a temporal knowledge graph (TKG) informer network for RUL prediction. A TKG of equipment health status integrates sensor data with structure data through prior knowledge so as to characterize various ST features accurately. The graph structure and node information (spatial-domain features) of the TKG at each moment are embedded in a low-dimensional temporal graph representation (TGR). An informer network extracts variable information (temporal-domain features) to generate TGR for RUL prediction. The proposed method was evaluated on public datasets and was found to achieve much higher performance than other state-of-the-art models. The TKG datasets are available at IEEE DataPort: https://dx.doi.org/10.21227/jgs2-kt12.**

be collected in a short time, providing an important foundation for RUL prediction. Hence, the use of monitoring data and knowledge for RUL prediction has become a research focus. RUL prediction methods can be roughly categorized as model-based [3], [4] or data-driven [5], [6]. Model-based methods use physical or mathematical models to describe system degradation processes. Although they may have high prediction accuracy, they cannot accurately model complex equipment of different categories; therefore, data-driven methods have become the mainstream technology [7].

Data-driven methods analyze monitoring data to obtain the mapping relationship between data and RUL [8] and can be divided into traditional machine learning (ML)-and deep learning (DL)-based methods. Traditional ML-based methods, such as support vector machine [9], multilayer perceptron (MLP) [10], radial basis function [11], and extreme learning machines [12], can model complex monitoring data but rely on extensive manual feature engineering, and their modeling capability is insufficient for massive data. DL-based methods can auto-

5.1 图数据库概述

- 利用知识图谱与大模型各自的优势相互赋能（1+1），并结合上层应用集成，实现两者技术的互补。
- 利用知识图谱间的互联互通及大模型间的集成调度（N+N），实现融合后系统能力的持续增强。



知识图谱与大模型融合技术框架

5.1 图数据库概述

2、图数据库的优势

1）图数据库不需要执行连接操作，查询速度快。

- 关系型数据库需要执行连接操作来进行关系的查找；
- 图数据库只需要沿着节点之间的边来查找。

2）建模过程较简单。

- 通过关系型数据库建立多对多的关系需要创建复杂的表格，如至少三个基本表。
- 图数据库对实体的关系建模非常方便。

3）可以为实体之间的多种关系进行建模。

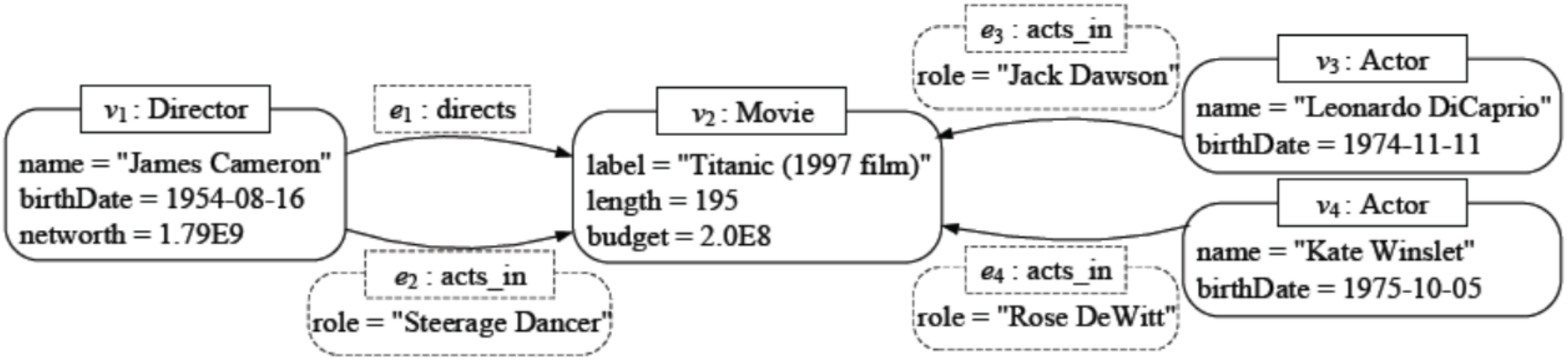
5.1 图数据库概述

3、属性图数据模型

图数据模型定义了图中节点以及关系的存储和实现方法。属性图是一种常用的图数据模型，被广泛采用，如Neo4j，其主要特点如下：

- 图由节点和关系（边）组成；
- 节点有一个或多个标签；
- 节点有一组属性（键值对）；
- 关系有一个类型，可以有一组属性；
- 关系是有方向的，只有一个开始节点和一个结束节点。

5.1 图数据库概述



- 节点={V1, V2, V3, V4}
- 关系={e1, e2, e3, e4}
- e1=(v1, v2), e2=(v1, v2)…
- 每个节点都有一个或多个标签和一组属性
- 每个关系都有一个类型和一组属性

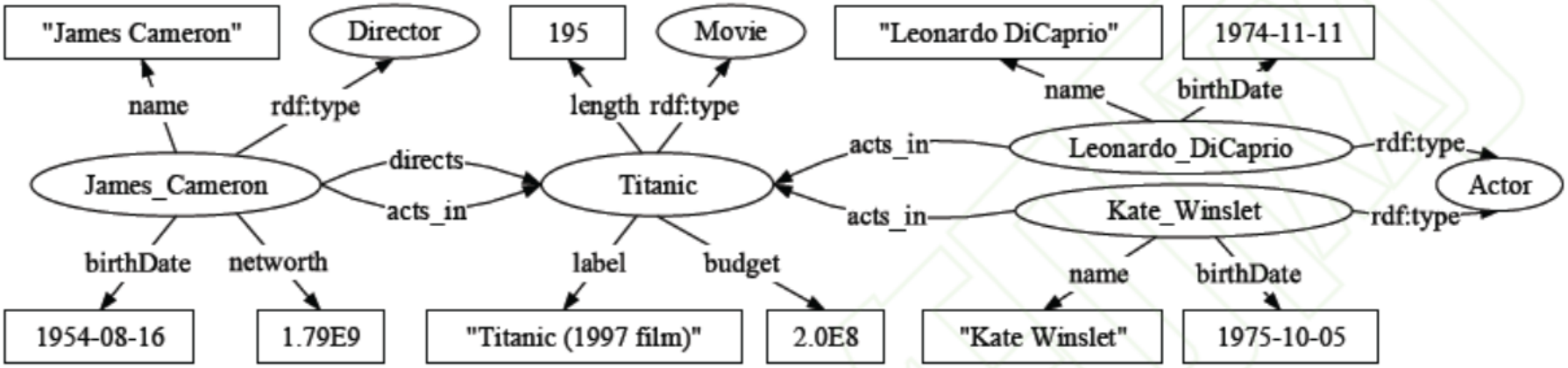
5.1 图数据库概述

4、三元组图数据模型

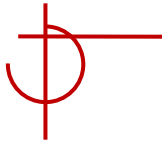
三元组模型是另外一种重要的图数据模型，包含主谓宾的数据结构，通过三元组来表达实体与实体之间的关系，或者属性与属性值之间的关系，有两种形式：

- （实体，关系，实体）
- （实体，属性，属性值）

5.1 图数据库概述



{ (James_Cameron, rdf:type, Director), (James_Cameron, name, "James Cameron"),
(James_Cameron, birthDate, 1954-08-16), (James_Cameron, networth, 1.79E9),
(James_Cameron, directs, Titanic), (James_Cameron, acts_in, Titanic),
(Titanic, rdf:type, Movie), (Titanic, label, "Titanic (1997 film)"),
(Titanic, budget, 2.0E8), (Titanic, length, 195),
(Leonardo_DiCaprio, rdf:type, Actor), (Leonardo_DiCaprio, name, "Leonardo DiCaprio"),
(Leonardo_DiCaprio, birthDate, 1974-11-11), (Leonardo_DiCaprio, acts_in, Titanic),
(Kate_Winslet, rdf:type, Actor), (Kate_Winslet, name, "Kate Winslet"),
(Kate_Winslet, birthDate, 1975-10-05), (Kate_Winslet, acts_in, Titanic) }.□



5.2 Neo4j的基本元素

Neo4j是一个Java开发的图数据库，适用于大量复杂、互连接、低结构化的数据，避免了由于连接操作产生的性能问题，其特点包括：

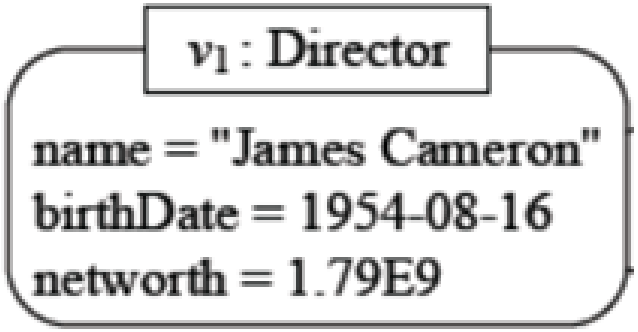
- 具有非常高效的查询性能。
- 提供了非常快的图算法、推荐系统和OLAP风格的分析。
- 提供了广泛使用的REST接口，能够方便地集成到基于JAVA、PHP、.NET和JavaScript的环境里。
- 高可用性，支持集群。
- 完整的ACID支持。

5.2 Neo4j的基本元素

1、节点（Node）

节点是图数据库中的一个基本元素，用来表示一个实体记录，类似于关系数据库中的一条记录，文档数据库中的一个文档。

- 每个节点可以有一个或多个标签；
- 每个节点可以有多个属性。



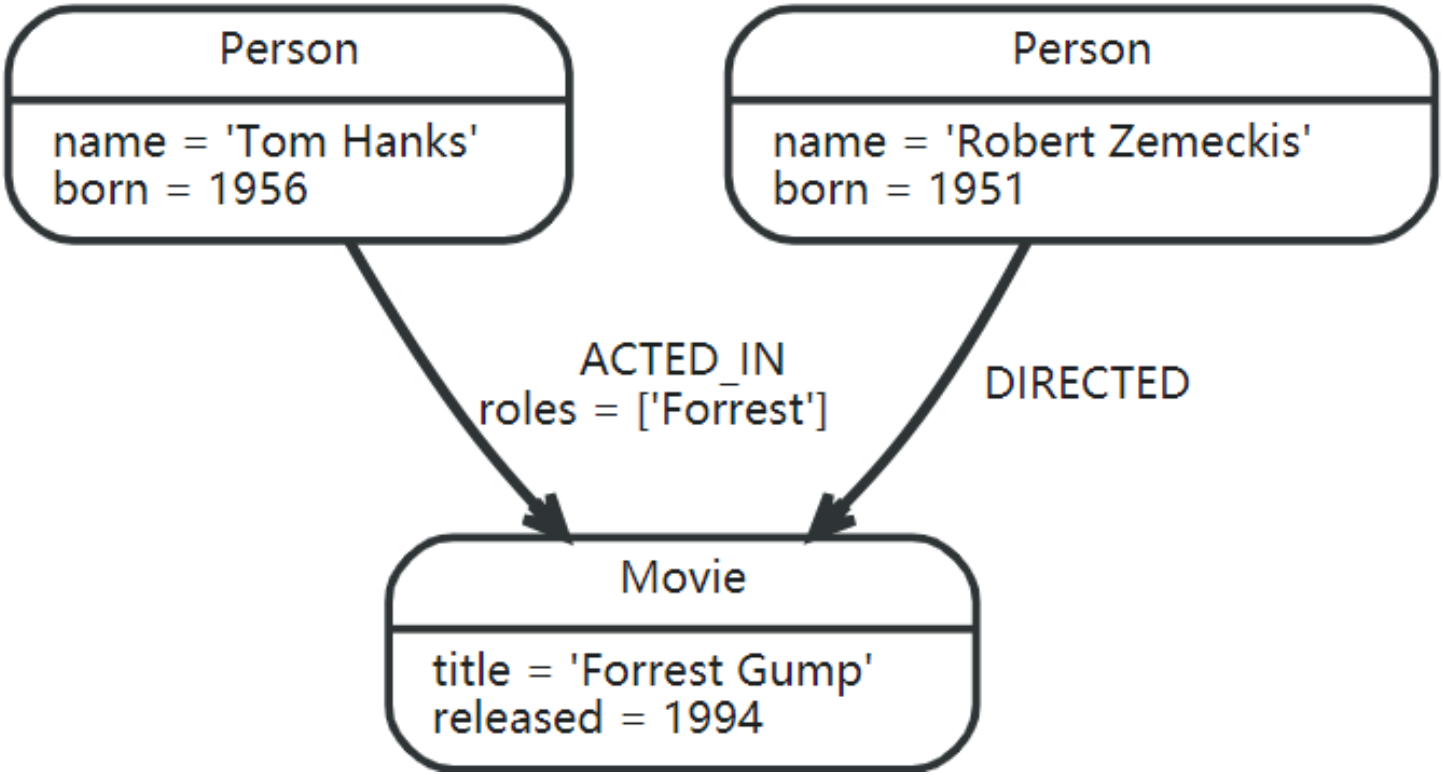
5.2 Neo4j的基本元素

2、关系（Relationship）

关系是图数据库中的一个基本元素，用边来表示，用来连接两个节点，表示节点之间的关系。

- 关系必须有一个开始节点和一个结束节点，且都不为空。
- 关系有方向。
- 关系只能有一个类型。
- 关系可以有多个属性。

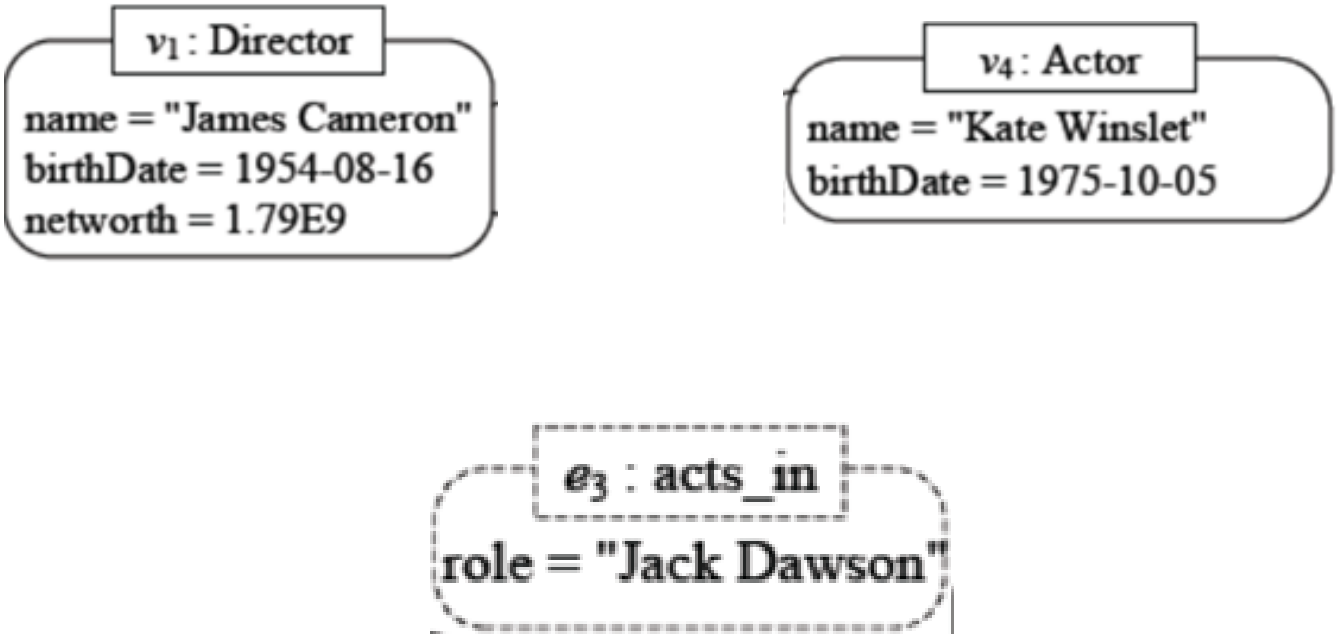
5.2 Neo4j的基本元素



5.2 Neo4j的基本元素

3、标签（Label）

标签定义了节点或关系的角色和类型。

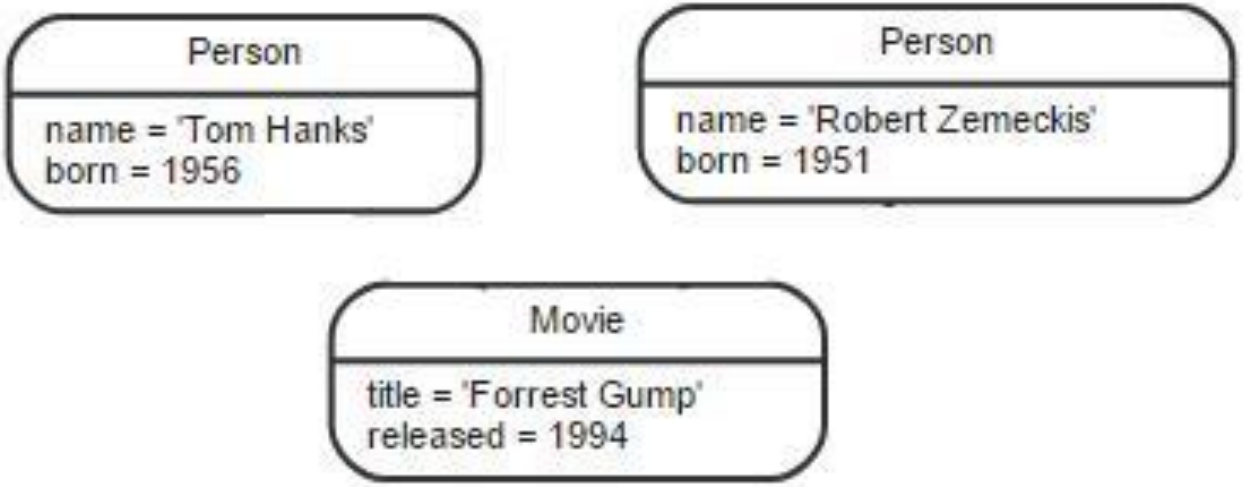


5.2 Neo4j的基本元素

4、属性（Attribute）

节点和关系都可以有属性，属性是由键值对组成的，即（属性，属性值），属性值可以是基本的数据类型，或者由基本数据类型组成的数组。

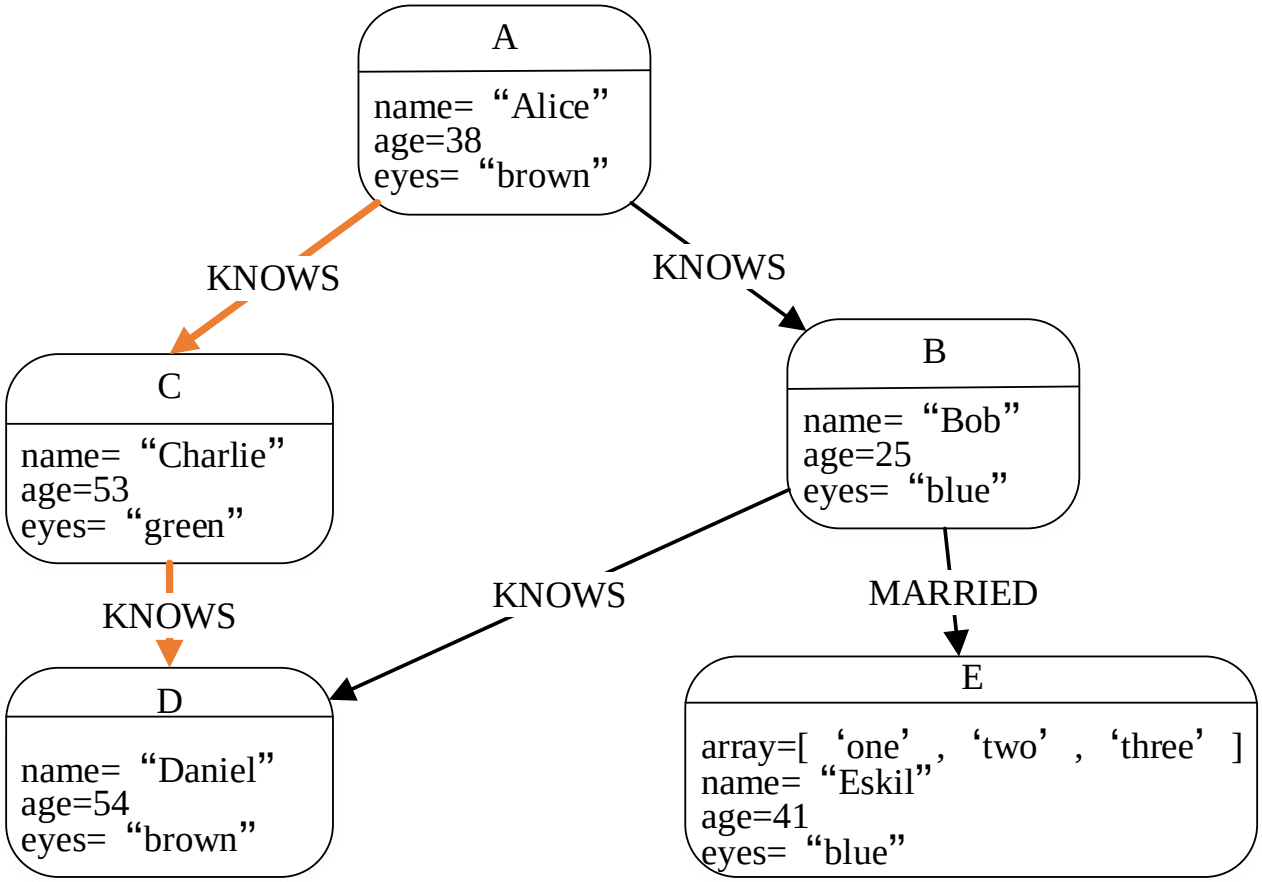
- 数值型（整性、浮点型）
- 布尔型
- 字符串
- 时间型



5.2 Neo4j的基本元素

5、路径（Path）

路径是由一些节点和一些关系组成的，路径经常是作为一个查询或者遍历的结果，规定一个路径至少一个节点。



5.2 Neo4j的基本元素

6、遍历（Traversal）

遍历是对图的一种操作，即按照一定的规则，根据图中节点之间的关系，以此访问所有相关联的节点。Neo4j提供了一套高效的遍历API，可以制定遍历的规则，并返回遍历的结果。

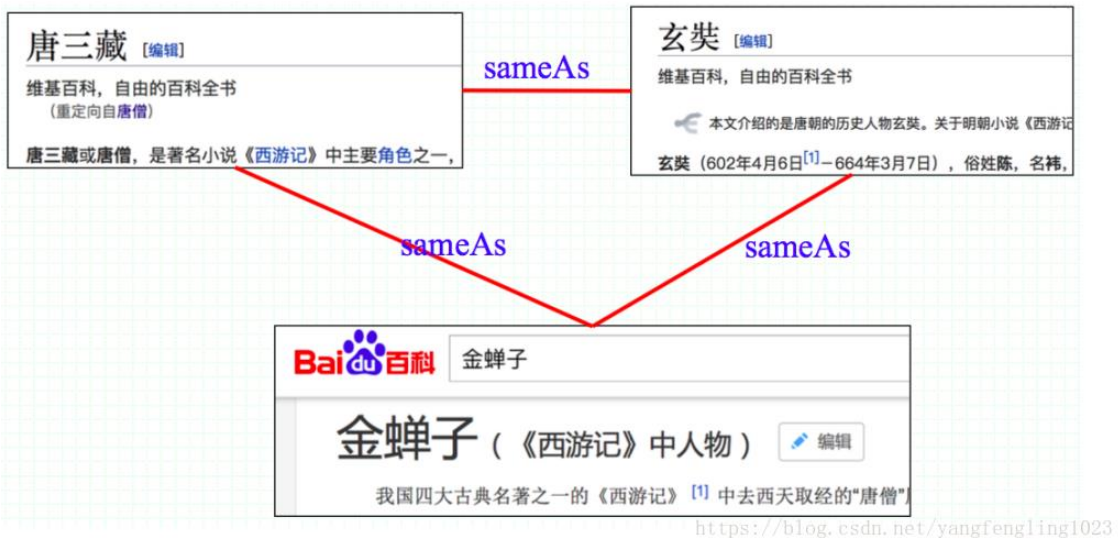
- 深度优先：沿着图的深度遍历节点，尽可能深的搜索图分支。
- 广度优先：从根节点开始，沿着图的宽度遍历图的节点
- 最短路径：用于计算一个节点到其他所有节点的最短路径。

5.2 Neo4j的基本元素

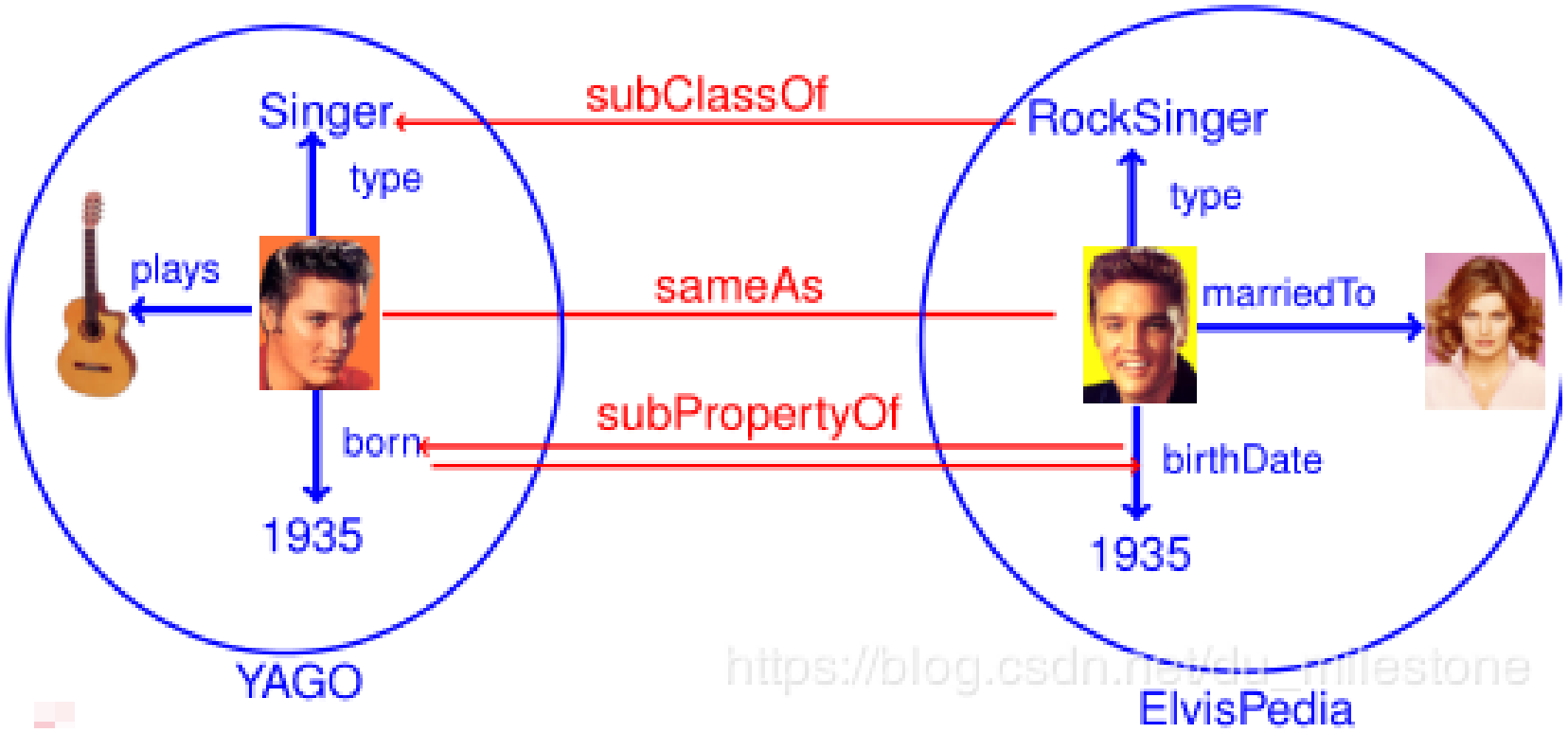
7、图融合（Fusion）

将多个图融合为一个图，发现两个图之间的等价节点、等价或为包含关系等概念或属性。

- 节点对齐
- 关系对齐
- 属性对齐



5.2 Neo4j的基本元素



5.3 Neo4j的基本操作

Cypher语言是一种声明式图查询语言，表达高效查询和更新图数据库，其语句可以分为三类：

1、写语句：对图数据库的节点、边、属性进行增删改操作。

2、读语句：对图数据库进行查询和统计操作。

3、通用语句：施加限制条件。

5.3 Neo4j的基本操作

为了方便引用图数据库操作过程中一部分结果，如节点、边、模式等，将这些结果赋值给一个变量。

例如：MATCH (n) RETURN n //查询整个图数据库

定义变量的规则：

① Cypher语言的关键字不区分大小写，但是变量、属性值，标签，关系类型和变量是区分大小写的。

② 必须以字母开头，可以包含下划线、字母和数字。

③ 变量命名规则也适用于属性的命名。

④ 变量仅在一个查询内可见，不能用于后续查询。

5.3 Neo4j的基本操作

1、 Cypher语言的写语句

1）Create:创建节点和边

2）Merge：匹配模式或创建模式

3）Set：更新节点的标签以及节点和关系的属性。

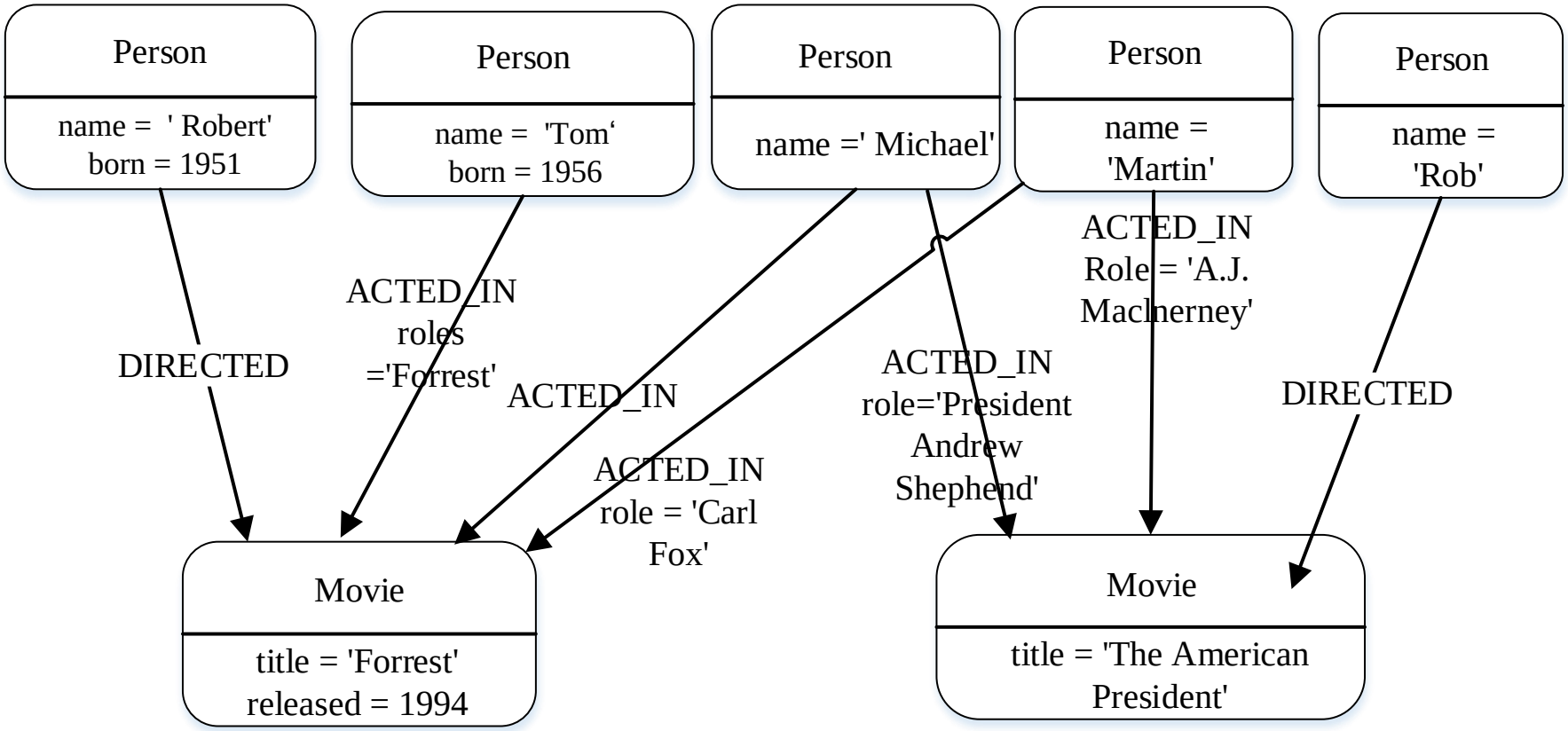
4）Delete：用于删除图元素（节点、关系或者路径）

5）Remove：移除节点的标签或者节点和关系的属性。

6）Foreach：更新列表中的节点和边数据

7）Create Unique：匹配模式，否则创建未匹配到的；在高版本中不再支持该命令。

5.3 Neo4j的基本操作



5.3 Neo4j的基本操作

1）创建节点Create语句

节点用一对小括号（）表示，是图数据库的一个节点，节点可以有一个或多个标签，可以有一个或多个属性。

Create（Variable:Lable1:Lable2 {Key1:Value1, Key2:Value2})

每个节点都有一个整数ID，在创建新的节点时，Neo4j自动为节点设置ID值，在整个数据库中，节点的ID值是递增的和唯一的。

5.3 Neo4j的基本操作

- 创建一个空节点

  create ()
- 创建并返回一个空节点

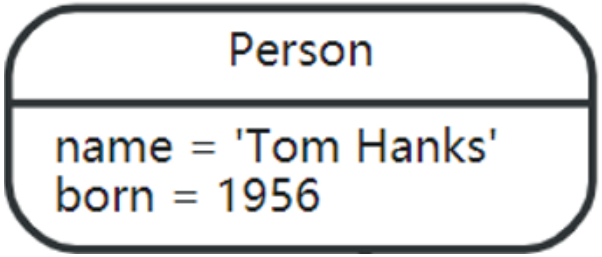
  create (n)

  return n
- 创建一个标签是Person的节点

  create

  (p:Person:Student { name: 'Tom Hanks', born: 1956 })

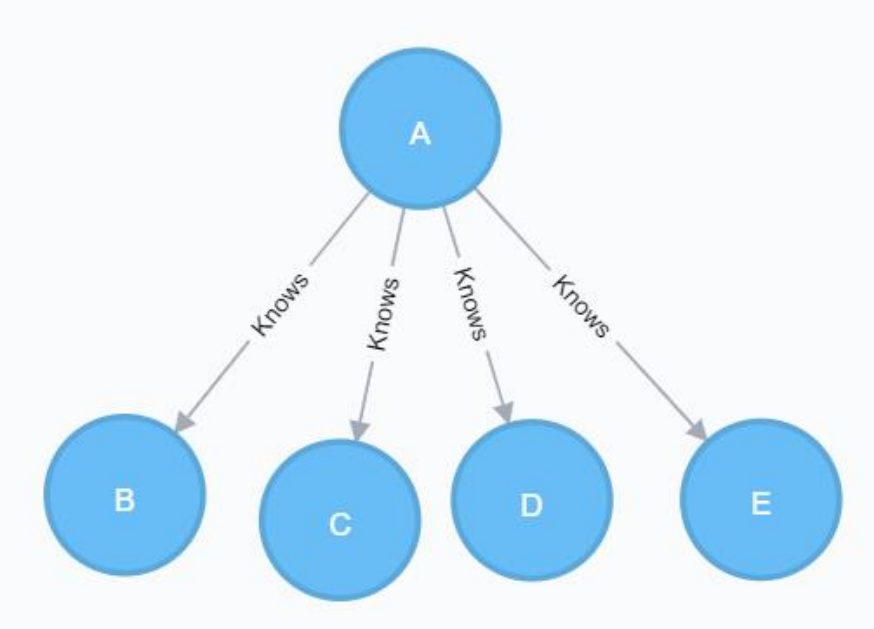
  return p



5.3 Neo4j的基本操作

- 创建5个人之间的认识关系

```
Create
(a:Person{name:'A', age:21}),
(b:Person{name:'B', age:22}),
(c:Person{name:'C', age:23}),
(d:Person{name:'D', age:24}),
(e:Person{name:'E'}),
(a)-[:Knows]->(b),
(a)-[:Knows]->(c),
(a)-[:Knows]->(d),
(a)-[:Knows]->(e)
```



5.3 Neo4j的基本操作

2）创建关系Create语句

关系用[]表示，创建关系时要指定头节点、尾节点和类型，并指定关系方向：

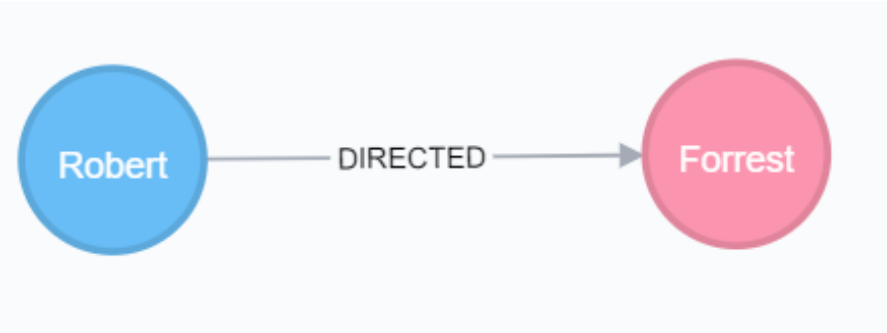
- 符号“（a）-[r:type]->（b）”，表示有方向的关系。
- 符号“（a）<-[r:type]-（b）”，表示有方向的关系。

StartNode-[Variable:Type {Key1: Value1, Key2:Value2}] -> EndNode

5.3 Neo4j的基本操作

- 创建两个节点和一个关系

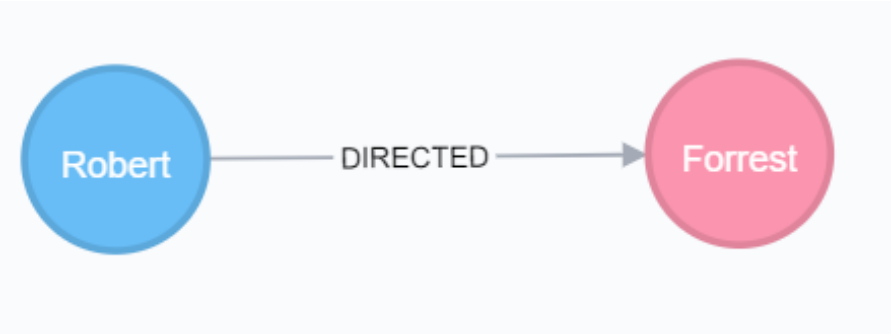
```
CREATE
(n:Person { name: 'Robert', born: 1951 })
-[r:DIRECTED] ->
(m:Movie { title: 'Forrest', released: 1951 })
RETURN  n,r,m
```



5.3 Neo4j的基本操作

- 创建两个节点和一个关系

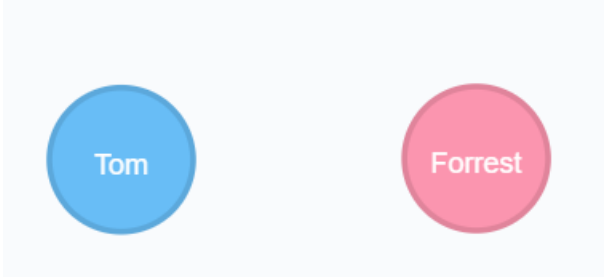
```
CREATE
(n:Person { name: 'Robert', born: 1951 }),
(m:Movie { title: 'Forrest', released: 1951 }),
(n)-[r:DIRECTED] ->(m)
RETURN  n,r,m
```



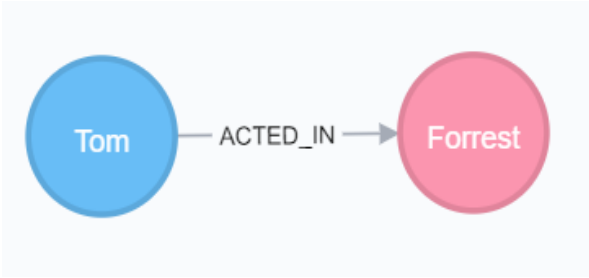
5.3 Neo4j的基本操作

- 为已存在的节点创建一个新关系

```
create  (:Person{name:'Tom', age:35}),
(:Movie{title:'Forrest', released:1994})
```




```
MATCH (a:Person),(b:Movie)
WHERE a.name = 'Tom' AND b.title =
'Forrest'
CREATE (a)-[r:ACTED_IN {
roles:'Forrest' }]->(b)
RETURN a,b
```



5.3 Neo4j的基本操作

- Create语句无法创建没有方向的关系

  Create (a:Person{name:'A'})-[:knows]-(b:Person {name: 'B'})-[:knows]-(c:Person {name: 'C'})

  RETURN a,b,c

  Error: Neo.ClientError.Statement.SyntaxError: Only directed relationships are supported in CREATE

5.3 Neo4j的基本操作

3）MERGE语句

MERGE语句的作用有两个：

- 若模式存在，则匹配指定的模式；
- 若模式不存在，则创建新的模式；
- 功能是Match子句和Create子句的组合。

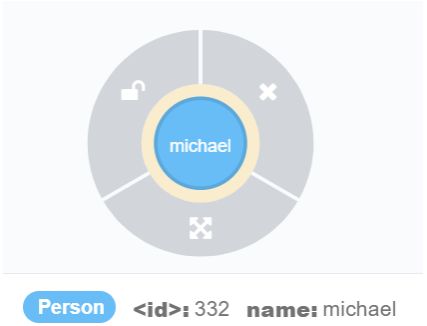
当在整个模式上使用MERGE时，要么整个模式匹配到，要么整个模式被创建。MERGE不能部分应用于模式

5.3 Neo4j的基本操作

- 匹配‘michael douglas’的节点，如果不存在该节点，则创建一个新节点

  merge(n:Person{name:'michael'})

  return n



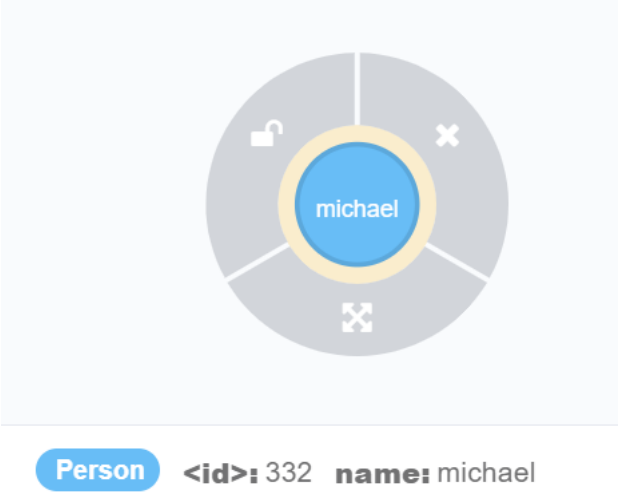
- 匹配‘Wang‘节点和’Gao Ming‘节点之间的关系，若不存在该关系，则重新创建两个节点（即使这两个节点存在）和一个新的关系。

  merge (s:Teacher{name:'Wang'})-[r:Directed]-(t:Student{name:'Gao Ming'}) return s, t

5.3 Neo4j的基本操作

- 检查‘michael’节点是否存在，若不存在该节点，则创建该节点并设置它的属性；若存在所匹配的节点，则不设置它的属性。

```
Merge (n:Person{name:'michael’})
on create set n.age=20
return n
```



5.3 Neo4j的基本操作

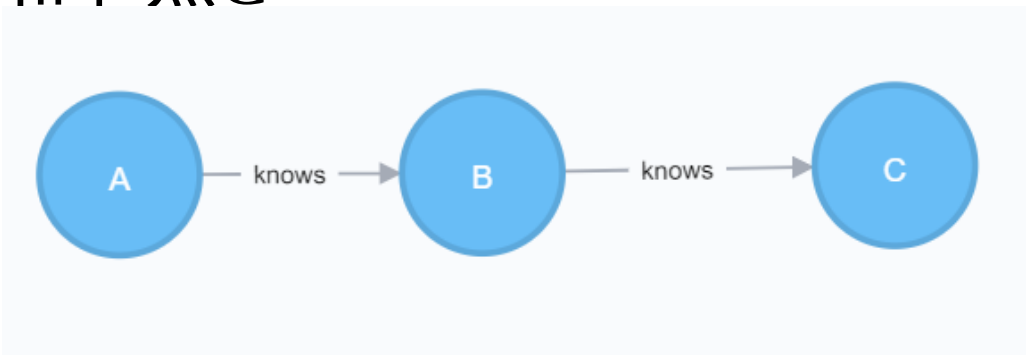
- Merge引用新创建的节点。

  Create (a:Person{name:’A'})

  Merge (a)-[:knows]->(b:Person {name: ‘B'})-[:knows]->(c:Person {name: ‘C'})

  RETURN a,b,c

  结果：Create语句创建了节点A，Merge语句创建了节点B和节点C



5.3 Neo4j的基本操作

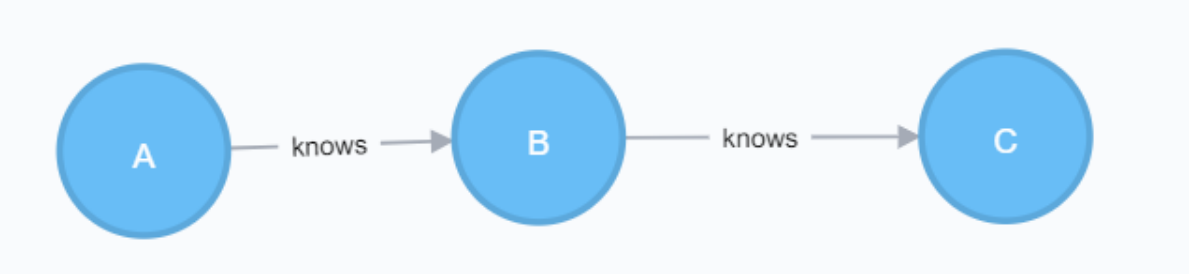
- Merge可省略关系的方向，但创建的关系有方向。

  Create (a:Person{name:'A'})

  Merge (a)-[:knows]-(b:Person {name: 'B'})-[:knows]-(c:Person {name: 'C'})

  RETURN a,b,c

  结果：Merge引用了节点A，创建了节点B和节点C。



5.3 Neo4j的基本操作

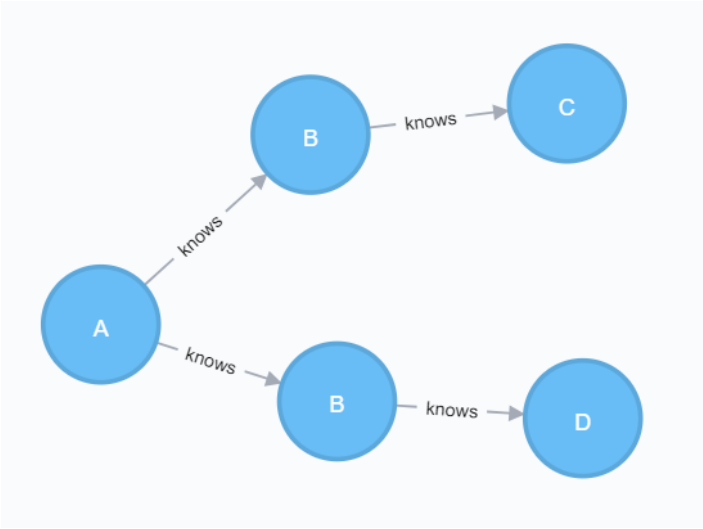
- Merge引用已有节点。

  Match (a:Person{name:’A'})

  Merge (a)-[:knows]->(b:Person {name: ‘B'})-[:knows]->(c:Person {name: ‘D'})

  RETURN a,b,c

  结果：Merge引用了节点A，又创建了节点B和节点D。



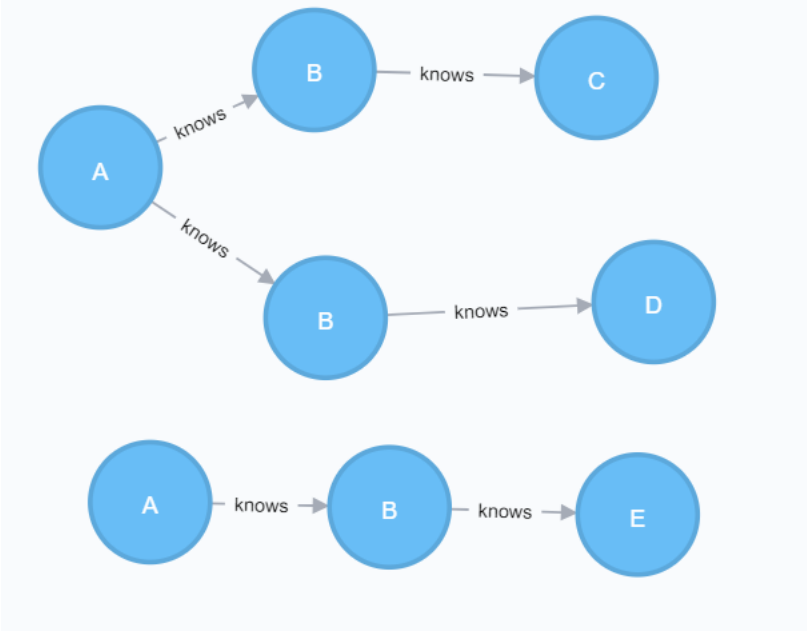
5.3 Neo4j的基本操作

- Merge不引用已有节点。

  Merge (a:Person{name:'A'})-[:knows]->(b:Person {name: 'B'})-[:knows]->(e:Person {name: 'E'})

  RETURN a,b,e

  结果：Merge语句重新创建了节点A、B和E。



5.3 Neo4j的基本操作

4）SET语句

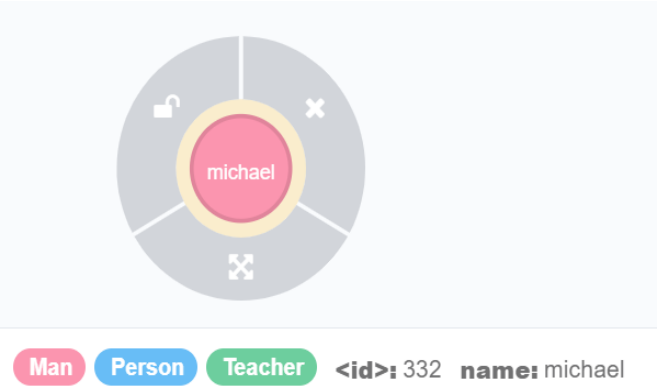
SET语句用于更新节点的标签以及节点和关系的属性。

- 为节点新增一个属性。

```
MATCH (n:Person{name:'michael'})
 SET n.born = 1955
 RETURN n
```

- 为节点新增两个标签。

```
MATCH (n:Person{name:'michael'})
SET n:Teacher:Man
RETURN n
```



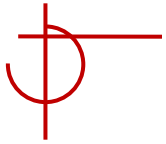
5.3 Neo4j的基本操作

- 将一个节点的属性值设置为Null，此时属性被删除

```
MATCH (n:Person{name:'michael'})
SET n.name=NULL
RETURN n
```

- 匹配节点，若匹配成功，则在该节点上设置属性；若匹配不成功，则创建新节点，不设置属性。

```
Merge(n:Teacher{name: 'michael’})
on match set n.title='Professor’
return n
```



5.3 Neo4j的基本操作

5）DELETE语句

DELETE语句用于删除图元素（节点、关系或者路径）。

- 不能只删除节点，而不删除与之相连的关系，要么显式地删除对应的关系。
- DETACH DELETE语句能够删除一个节点及其所有关系。

5.3 Neo4j的基本操作

- 删除一个没有关系的节点。

```
MATCH (n{name:'michael'})
DELETE n
```

- 删除一个带关系的节点。

```
MATCH (n{name:'michael'})
DETACH DELETE n
```

- 删除id号是269的节点。

```
MATCH(n) where  ID(n)=269  DELETE  n
```

- 删除所有节点及其关系。

```
MATCH(n)  DETACH DELETE  n
```

5.3 Neo4j的基本操作

6）REMOVE语句

REMOVE语句用于移除节点的标签或者节点和关系的属性。

- 移除节点的一个属性。

```
MATCH (n:Person{name:'michael'})
Remove n.born
RETURN n
```

- 移除节点的一个（或多个）标签。

```
Match(n:Person{name:'Cheng long'})
remove n:Person
return n
```

5.3 Neo4j的基本操作

7）FOREACH语句

FOREACH语句能够更新列表、路径等集合中的每一个元素，可执行的更新命令包括CREATE、SET、CREATE UNIQUE、DELETE。

语法格式：

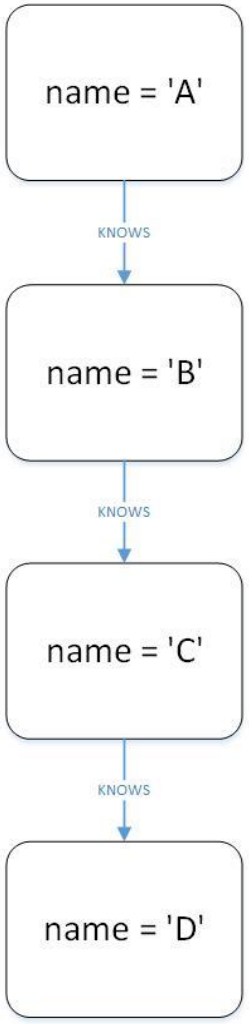
FOREACH (变量 IN [列表] | 更新语句)

5.3 Neo4j的基本操作

- 将路径上所有的节点的marked属性设为true，如果没有该属性，则新增一个markded属性。

```
create (a:Person{name:'A'}), (b:Person{name:'B'}),
(c:Person{name:'C'}), (d:Person{name:'D'}),
(a)-[:Know]->(b)-[:Know]->(c)-[:Know]->(d)
```

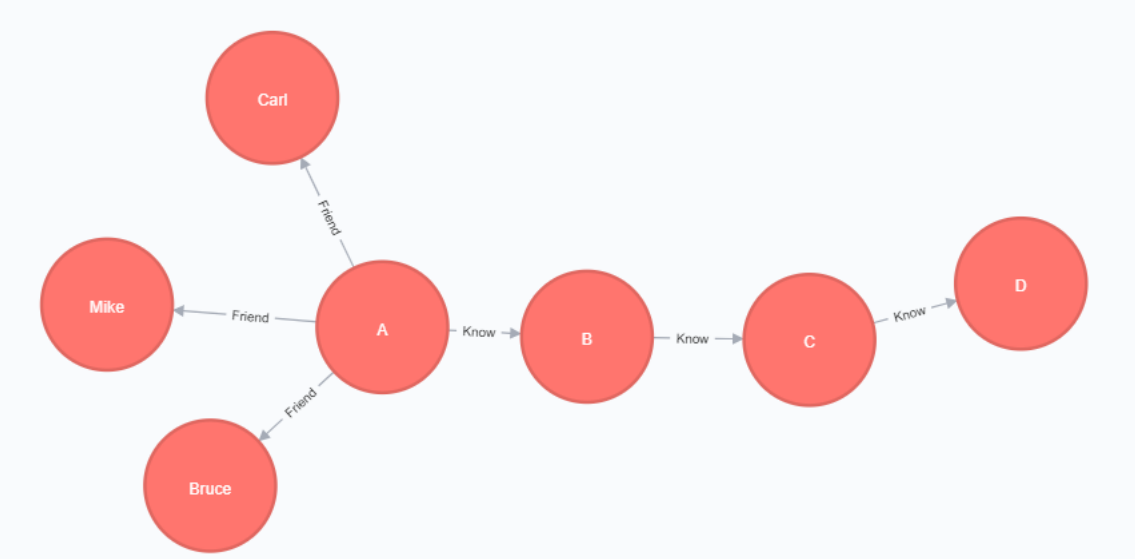
```
MATCH p=(begin)-[*]->(end)
WHERE begin.name = 'A' AND end.name= 'D'
FOREACH (n IN nodes(p) | SET n.marked=TRUE)
```



5.3 Neo4j的基本操作

- 将列表中的人全部加为‘A’的朋友

```
match(a:Person{name:'A'})
Foreach(n IN ['Mike',"Carl","Bruce"] | create (a)-[:Friend]->(:Person{name:n}))
```



5.3 Neo4j的基本操作

8）CREATE UNIQUE语句（高版本不支持）

- CREATE UNIQUE语句相当于MATCH和CREATE的混合体，其只创建未匹配成功的节点和关系，需要与MATCH语句配合使用：
  - CREATE UNIQUE语句不能单独创建节点与关系，需要配合Match语句或Create语句。
  - 如果CREATE UNIQUE语句所有创建的节点和关系都存在，则匹配它们。
  - 如果CREATE UNIQUE语句所要创建的某些节点和关系不存在，则创建它们；
- MERGE语句只要不是完全匹配，则创建所有节点和关系。
- CREATE UNIQUE语句只创建未匹配到的。

5.3 Neo4j的基本操作

- 无法单独创建新节点。

  CREATE UNIQUE (n { name:"X" }) RETURN n;

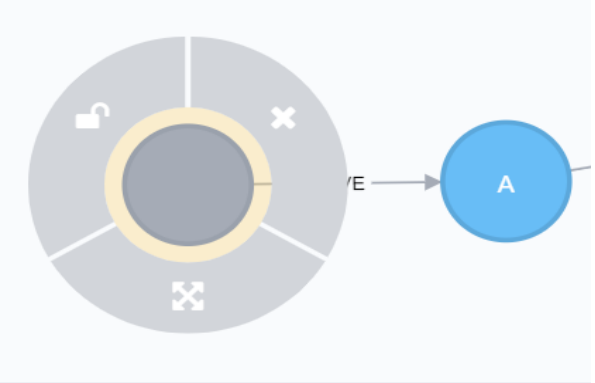
  Error： Neo.ClientError.Statement.SemanticError: This pattern is not supported for CREATE UNIQUE

- 需要配合Match语句使用。

  MATCH (root { name: 'A' })

  CREATE UNIQUE (root)-[:LOVE]-(someone{name: 'Alice'})

  RETURN someone



5.3 Neo4j的基本操作

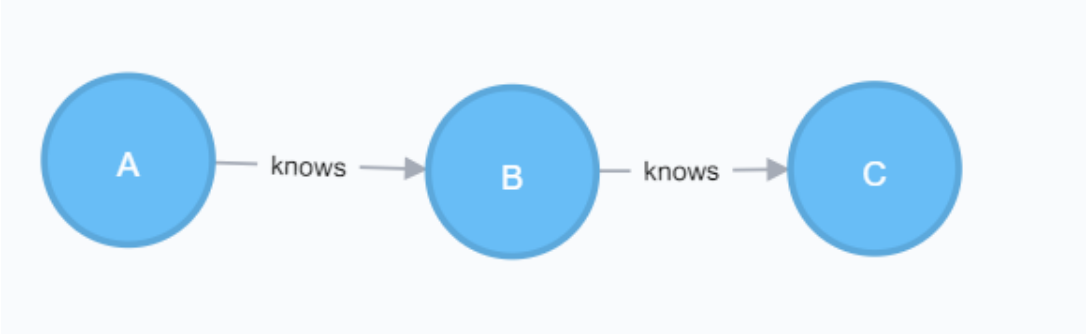
- CREATE UNIQUE引用已有节点。

  Create (a:Person{name:'A'})

  CREATE UNIQUE (a)-[:knows]->(b:Person {name: 'B'})-[:knows]->(c:Person {name: 'C'})

  RETURN a,b,c

  结果： CREATE UNIQUE创建了2个节点B、C



5.3 Neo4j的基本操作

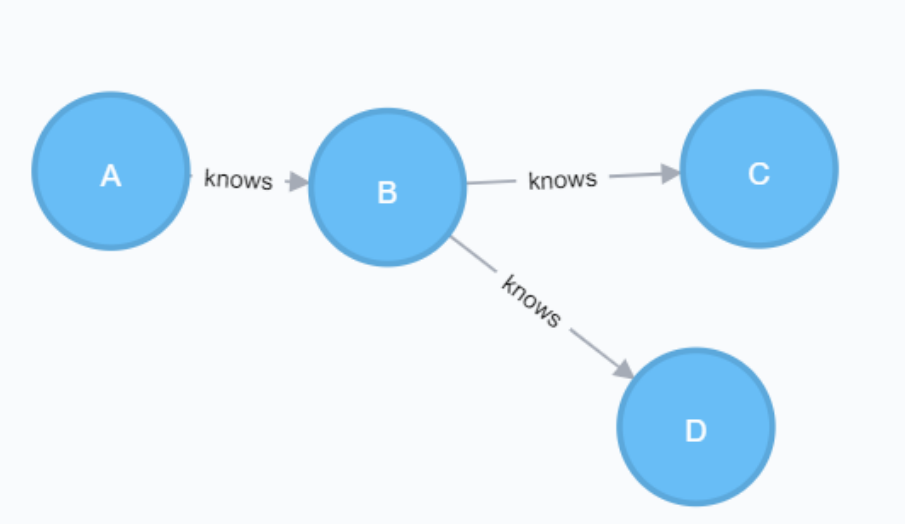
- CREATE UNIQUE引用已有节点。

  Match (a:Person{name:'A'})

  CREATE UNIQUE (a)-[:knows]->(b:Person {name: 'B'})-[:knows]->(d:Person {name: 'D'})

  RETURN a,b,d

  结果： CREATE UNIQUE创建了2个节点B、D



5.3 Neo4j的基本操作

- CREATE UNIQUE可以省略关系的方向。

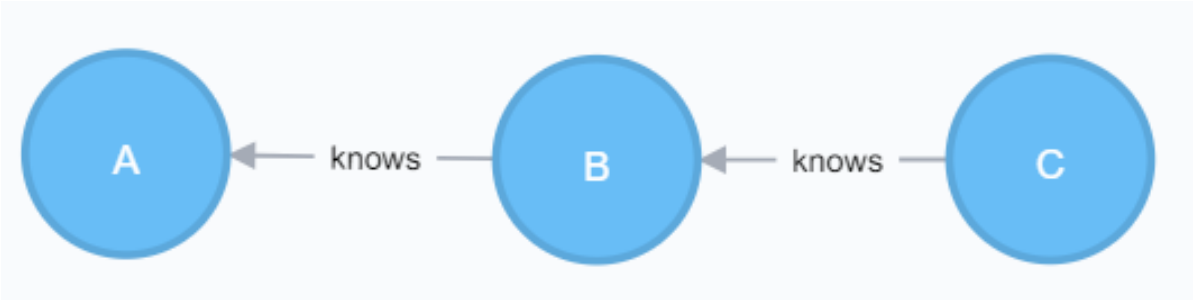
Create (a:Person{name:'A'})

CREATE UNIQUE (a)-[:knows]-(b:Person {name: 'B'})-[:knows]-(c:Person {name: 'C'})

关系可以不加方向

RETURN a,b,c

结果： CREATE UNIQUE创建了2个节点B、C



5.3 Neo4j的基本操作

2、 Cypher语言的读语句

1）Match ：根据指定的模式检索图数据库

2）Optional Match：根据指定的模式检索图数据，如果没有匹配到，将用null作为未匹配到部分的值。

3）Where：提供筛选条件。

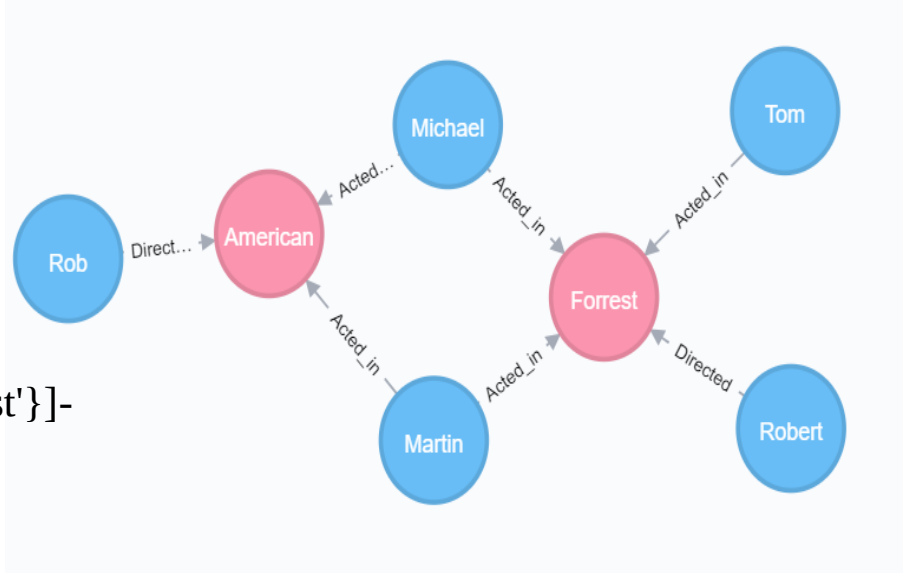
4）Aggregation：聚合数据。

5.3 Neo4j的基本操作

1）Match语句

Match语句用于根据指定的模式检索图数据库，where子句为Match模式增加条件约束。

```
create
(r:Person{name:'Robert', age:35}),
(t:Person{name:'Tom', age:56}),
(mi:Person{name:'Michael', age:55}),
(ma:Person{name:'Martin', age:20}),
(rob:Person{name:'Rob', age:42}),
(f:Movie{title:'Forrest', released:1994}),
(a:Movie{title:'American', released:2008}),
 (r)-[:Directed]->(f), (t)-[:Acted_in{role:'Forrest'}]-
>(f),
(mi)-[:Acted_in]->(f),
(ma)-[:Acted_in{role:'Carl Fox'}]->(f),
(mi)-[:Acted_in{role:'President'}]->(a),
(ma)-[:Acted_in{role:'A. J. Maclnerney'}]->(a),
(rob)-[:Directed]->(a)
```



5.3 Neo4j的基本操作

- 查询整个图数据库，返回图中所有的节点：

```
match （n）
return  n
```

- 查询返回age属性小于等于35的所有节点。

```
match(n)
where n.age<=35
return n
```

- 查询标签类型是Movie的所有节点。

```
match(n:Movie)
return n
```

- 查询具有指定属性值的节点

```
match(n{name:'Tom'})
return n
```

5.3 Neo4j的基本操作

- 查询与给定节点有关系的节点

```
match (n)--(m:Movie{title:'American'})
return n,m
```

- 查询有向关系的节点

```
match (n:Person{ name: 'Tom'})-->(movie)
return movie
```

5.3 Neo4j的基本操作

2）OPTIONAL MATCH语句

OPTIONAL MATCH匹配模式与Match类似，不同之处在于，如果没有匹配到，OPTIONAL MATCH将用NULL作为未匹配到部分的值。

- 查询指向指定节点的其他节点。

```
OPTIONAL MATCH (:Movie{ title:'Forrest' }) --> (x)
RETURN x
```

x
null

5.3 Neo4j的基本操作

- 查询电影“Forrest”节点所指向的节点。

```
MATCH (a:Movie{title:'Forrest'})
OPTIONAL MATCH (a) -[r:Acted_in]->(b)
RETURN r
```

返回了null关系，因为节点a没有Acted_in的外向关系。

r
null

5.3 Neo4j的基本操作

3）WHERE子句

WHERE为其他一些命令提供条件约束，不能单独使用。

- 查询age属性小于40的节点。

```
match(n)
where n.age<40
return n
```

5.3 Neo4j的基本操作

4）Aggregation语句

Cyper支持使用聚合来计算聚在一起的数据，聚合函数有多个输入值，然后基于它们计算出一个聚合值。例如avg函数计算多个数值的平均值。Min函数用于找到一组值中最小的那个值。

- 查询数据库中节点总数。

```
match (n) return count(*)
```

5.3 Neo4j的基本操作

- 简单计算所有值之和。

  match (n:Person)

  return sum(n.age)

- 查找数值列中的最小值。

  match (n:Person)

  return min(n.age)

- 将所有的值收集起来放入一个列表。

  match (n:Person) return collect(n.age)

collect(n.age)
[35, 56, 55, 20, 42]

5.3 Neo4j的基本操作

3、 Cypher语言的通用语句

1）Return：返回查询结果集中的内容。

2）Order by：对输出结果进行排序。

3）Limit：限制输出的行数。

4）Skip：跳过开始的一部分结果。

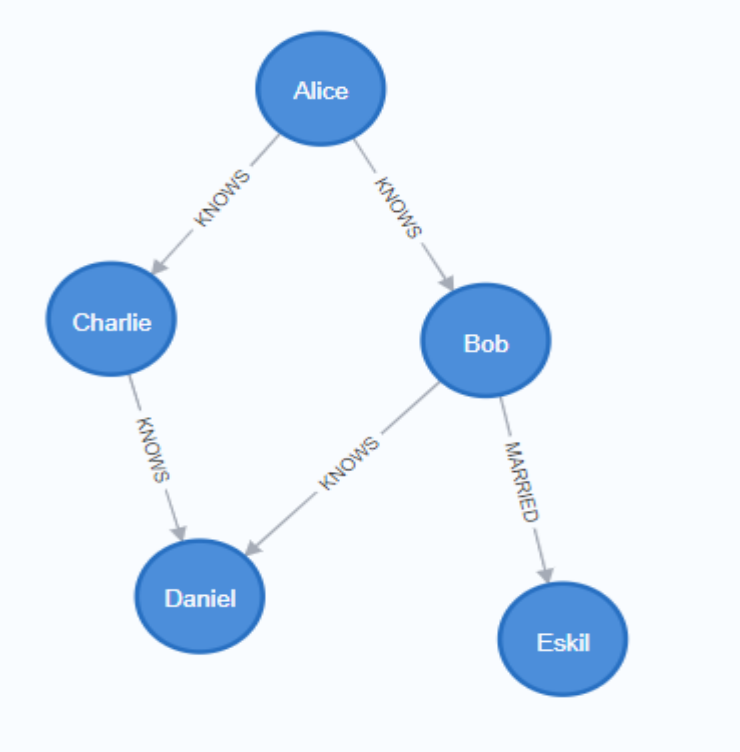
5）With：将分段的查询部分连接在一起，查询结果从一部分以管道形式传递给另外一部分作为开始点。

6）Unwind：将一个列表展开为一个行的序列，列表可以以参数的形式传入。

7）Union：将多个查询结果组合起来。

5.4 Neo4j的函数

```
create
(A:Person {name:'Alice',eyes:"brown",age:38}),
(B:Person {name:"Bob", eyes:"blue", age:25}),
(C:Person {name:"Charlie", eyes:'green',age:53}),
(D:Person {name:"Daniel", eyes:'brown', age:33}),
(E:Person{name:'Eskil',eyes:"blue",age:41,array:['one','two','three']}),
(A)-[:KNOWS]->(B),
(A)-[:KNOWS]->(C),
(B)-[:KNOWS]->(D),
(C)-[:KNOWS]->(D)
,(B)-[:MARRIED]->(E)
```



5.3 Neo4j的基本操作

1）Return语句

Return语句定义了查询结果集中返回的内容。返回的内容可以是路径、节点、关系或属性。

- 查找每个人的年龄

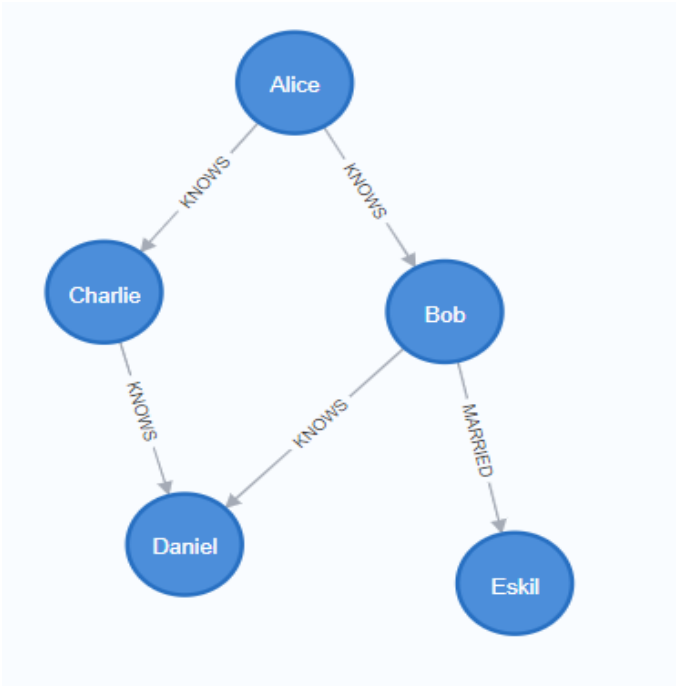
```
match (n:Person)
return n.age
```

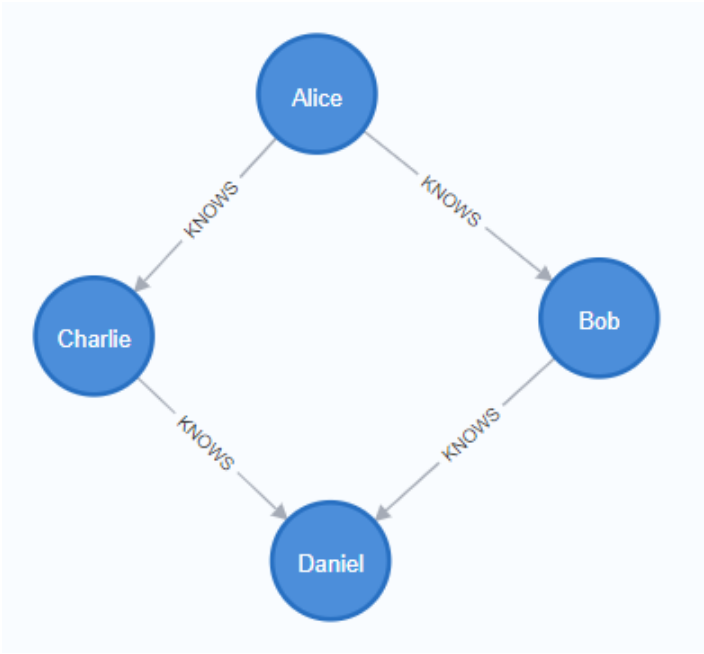
"n.age"
53
33
41
38
25

5.3 Neo4j的基本操作

- 查询Alice和Daniel之间长度是2的路径。

```
MATCH p=(a)-[*2]->(b)
WHERE a.name='Alice' AND b.name='Daniel'
RETURN p
```





5.3 Neo4j的基本操作

2）ORDER BY语句

ORDER BY语句用于对输出的结果进行排序，要紧跟在MATCH、RETURN或WITH的后面，它指定了输出的结果应该如何排序，ASC升序，DESC降序。

- 对输出进行排序。

```
MATCH (n:Person)
RETURN n.name,n.age
ORDER BY n.age desc
```

n.name	n.age
"Charlie"	53
"Eskil"	41
"Alice"	38
"Daniel"	33
"Bob"	25

5.3 Neo4j的基本操作

3）LIMIT语句

LIMIT限制输出的行数，可接受结果为正整数的表达式。

- 返回依据名字排序的三行输出

```
MATCH (n:Person)
RETURN n.name,n.age
ORDER BY n.age ASC
LIMIT 3
```

n.name	n.age
"Bob"	25
"Daniel"	33
"Alice"	38

5.3 Neo4j的基本操作

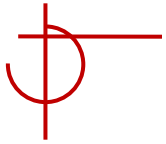
4）SKIP语句

SKIP定义了从哪行开始返回结果，使用SKIP可以跳过开始的一部分结果。

- 从第4个开始返回结果。

```
MATCH (n:Person)
RETURN n.name,n.age
ORDER BY n.age ASC
SKIP 3
```

n.name	n.age
"Eskil"	41
"Charlie"	53



5.3 Neo4j的基本操作

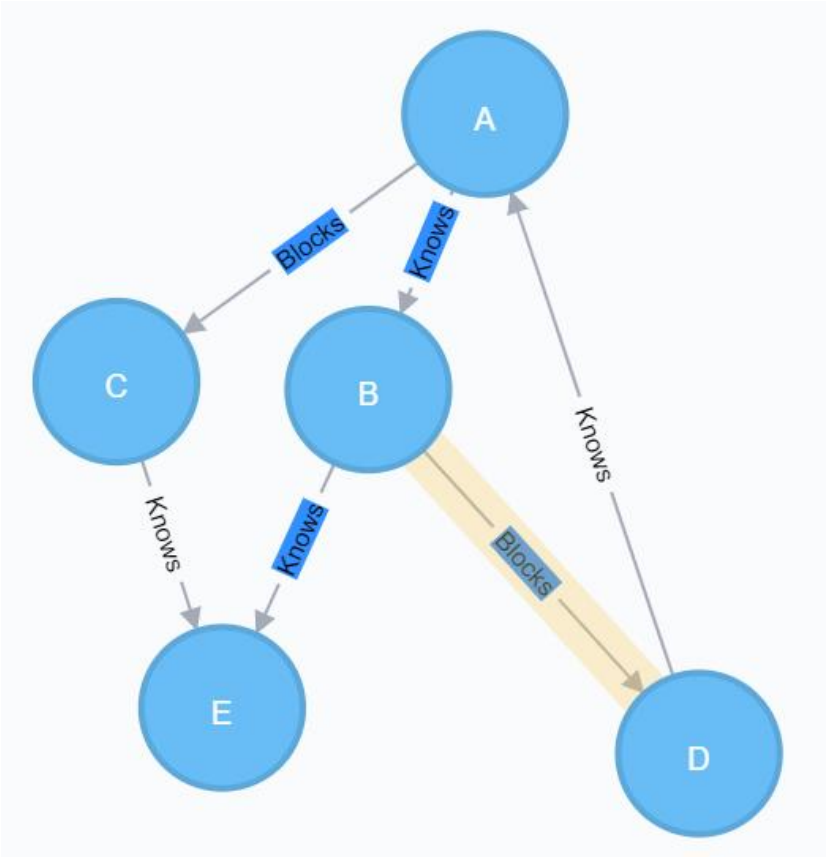
5）WITH语句

一个复杂的查询语句可能需要经多次处理。

WITH语句把上一个查询语句的结果作为输入，经适当处理，再把结果传递到后面的语句中。类似于管道

5.3 Neo4j的基本操作

- 创建以下5个人之间的关系。



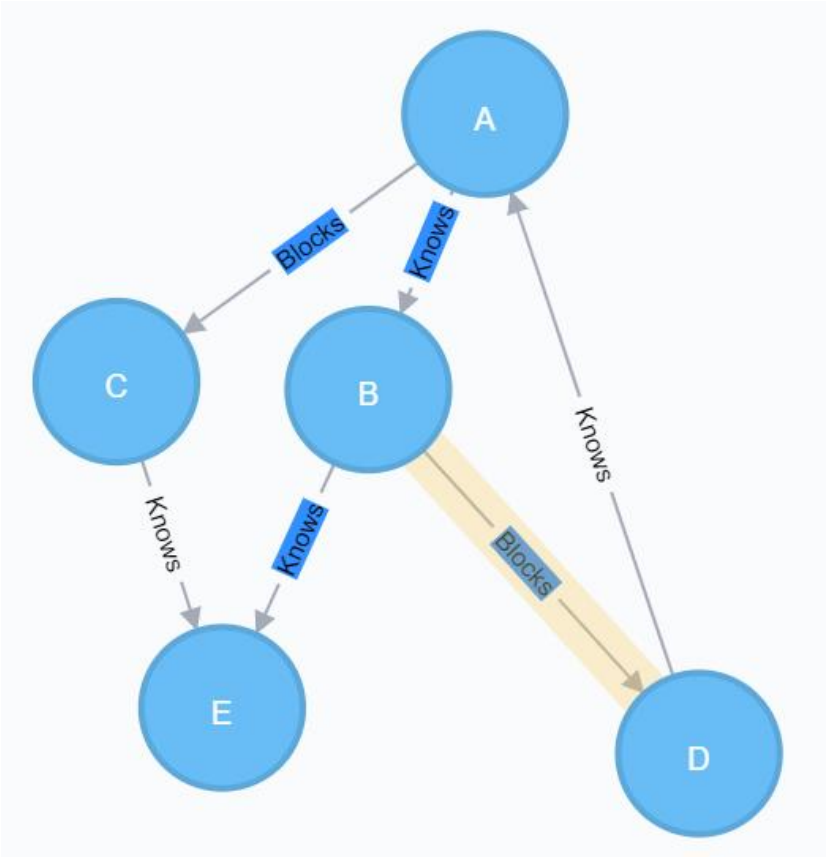

```
create
(a:Person{name:'A', age:21}),
(b:Person{name:'B', age:22}),
(c:Person{name:'C', age:23}),
(d:Person{name:'D', age:24}),
(e:Person{name:'E', age:25}),
(a)-[:Blocks]->(c),
(a)-[:Knows]->(b),
(b)-[:Knows]->(e),
(b)-[:Blocks]->(d),
(c)-[:Knows]->(e),
(d)-[:Knows]->(a)
Return a,b,c,d,e
```

5.3 Neo4j的基本

o.name
"B"
"A"

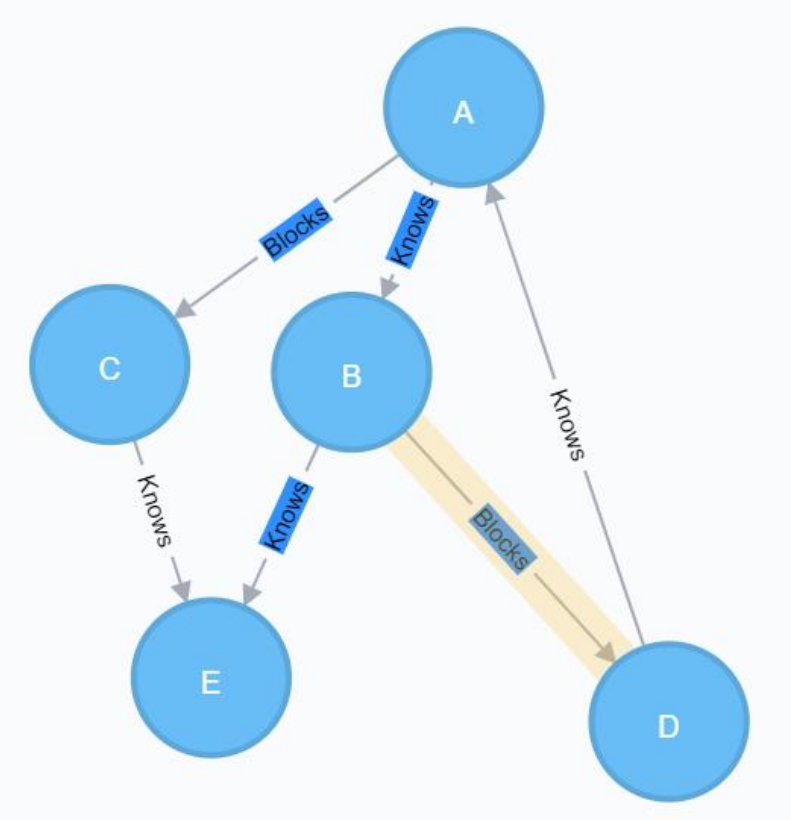
- 复杂查询语句。

```
MATCH (n { name: 'A' })--(m)
WITH m
ORDER BY m.name DESC
LIMIT 1
MATCH (m)--(o)
RETURN o.name
```



5.3 Neo4j的基本操作

- 在上述5人关系中，查询与D相连的人，且该人至少有2个外向关系




```
MATCH (d{name:'D'})--(n1)-->(n2)
return n1.name, n2.name
```

查询结果：

n1.name	n2.name
"B"	"E"
"A"	"C"
"A"	"B"

5.3 Neo4j的基本操作

```
MATCH (d{name:'D'})--(n1)-->(n2)
WITH n1, count(n2) AS num
WHERE num >1
RETURN n1.name,num
```

n1.name	num
"A"	2

聚合的结果必须通过with子句才能被过滤，即with子句保留n1，并新增聚合查询count(*)，通过where子句过滤。

5.3 Neo4j的基本操作

- 按照姓名排序后，收集姓名的列表

```
MATCH (n)
WITH n
ORDER BY n.name DESC
RETURN collect(n.name)  //输出列表
```

Neo4j高版本中，ORDER BY子句也可以跟在Match语句后面

```
MATCH (n)
ORDER BY n.name DESC
RETURN collect(n.name)
```

5.3 Neo4j的基本操作

6）UNWIND语句

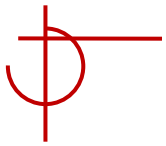
UNWIND语句将一个列表展开为一个行的序列，列表可以以参数的形式传入。

- 将原列表中的每个值以单独的行返回。

```
match (n:Person)
with collect(n.age) as x
UNWIND x as y
return y
```

x
[23, 24, 25, 21, 22]

y
23
24
25
21
22



5.3 Neo4j的基本操作

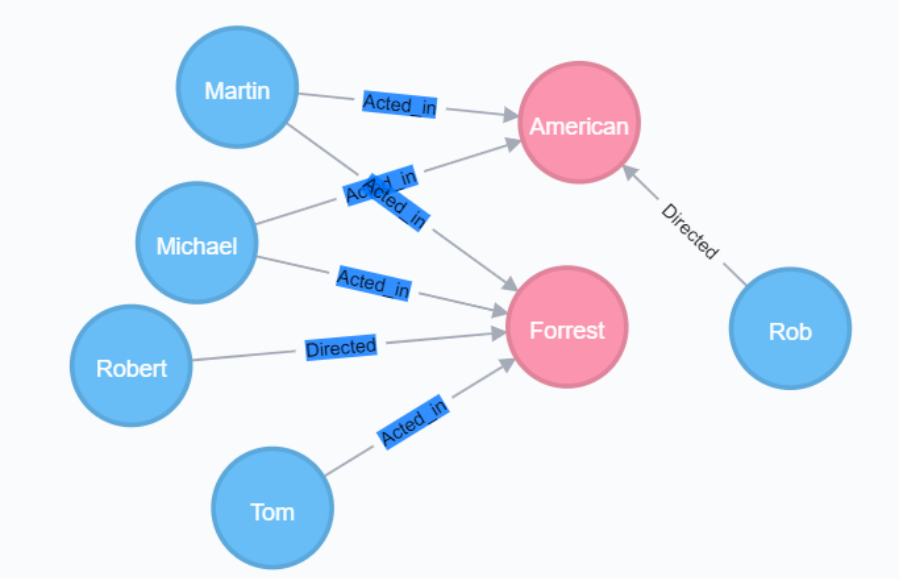
7）UNION语句

- UNION语句：将两个结果集合进行合并，去掉重复的元素。
- UNION ALL语句：将两个结果集合进行合并，不去掉重复的元素。

5.3 Neo4j的基本操作

- 用UNION ALL将两个查询的结果组合在一起。

```
MATCH (n:Person)
RETURN n.name AS name
UNION ALL
MATCH (n:Movie)
RETURN n.title AS name
```



name
"Robert"
"Tom"
"Michael"
"Martin"
"Rob"
"Forrest"
"American"

5.4 Neo4j的函数

1、断言函数

断言是一个布尔函数，即对给定的输入返回true或false，用于过滤子图。

- All：判断断言是否适用于列表中的所有元素。
- Any：判断断言是否至少适用于列表中的一个元素。
- None：断言不适用于列表中的任何元素，则返回true。
- Single：断言刚好只适用于列表中的某一个元素，则返回true。
- Exists：数据库存在该模式或节点中存在该属性时，返回true。

5.4 Neo4j的函数

1）ALL函数

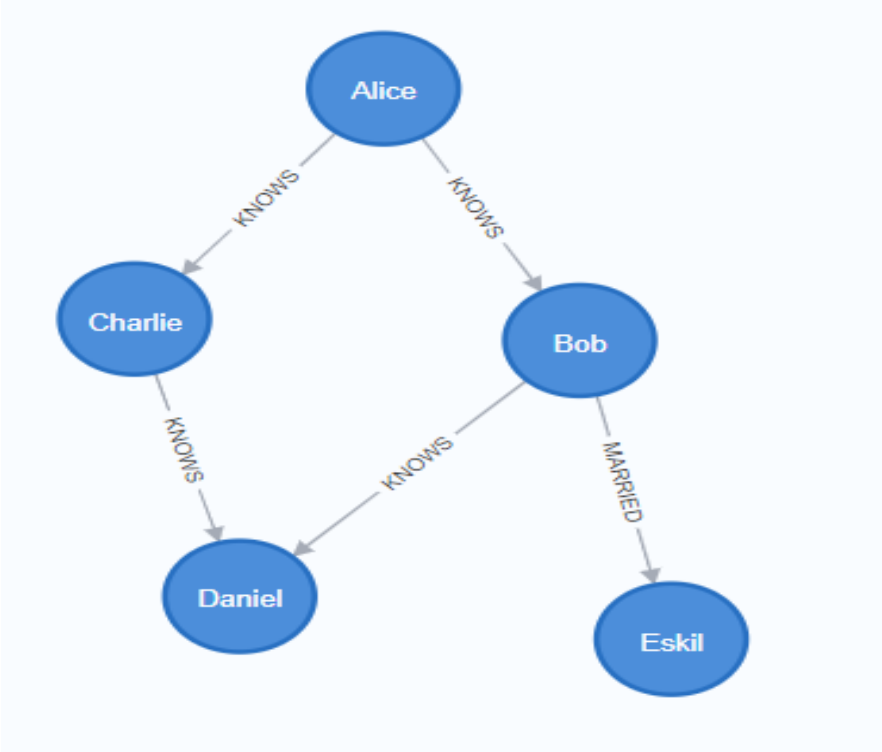
判断一个断言是否适用于列表中的所有元素。

All (variable IN list WHERE predicate)

- variable：断言变量
- list：列表表达式
- predicate：测试列表中所有元素的断言

5.4 Neo4j的函数

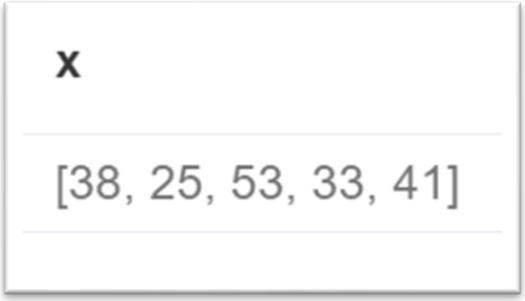
```
create
(A:Person {name:'Alice',eyes:"brown",age:38}),
(B:Person {name:"Bob", eyes:"blue", age:25}),
(C:Person {name:"Charlie", eyes:'green',age:53}),
(D:Person {name:"Daniel", eyes:'brown', age:33}),
(E:Person{name:'Eskil',eyes:"blue",age:41,array:['one','two','three']}),
(A)-[:KNOWS]->(B),
(A)-[:KNOWS]->(C),
(B)-[:KNOWS]->(D),
(C)-[:KNOWS]->(D)
,(B)-[:MARRIED]->(E)
```



5.4 Neo4j的函数

- 查询所有的Person节点，如果人的age都大于20，则返回。

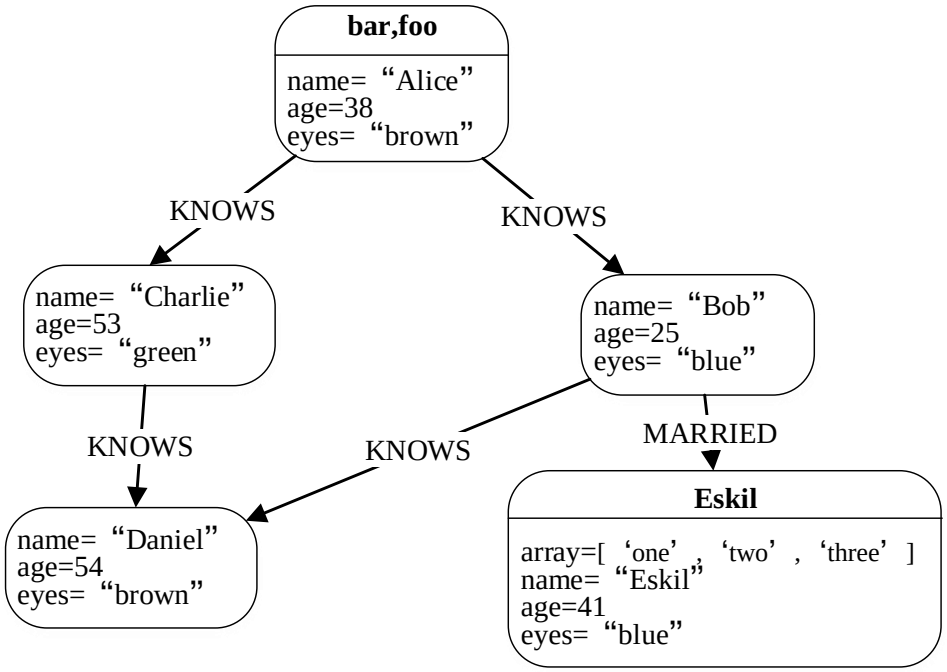
```
MATCH (a:Person)
WITH collect(a.age) as x
WHERE all(b IN x WHERE b>20)
return x
```

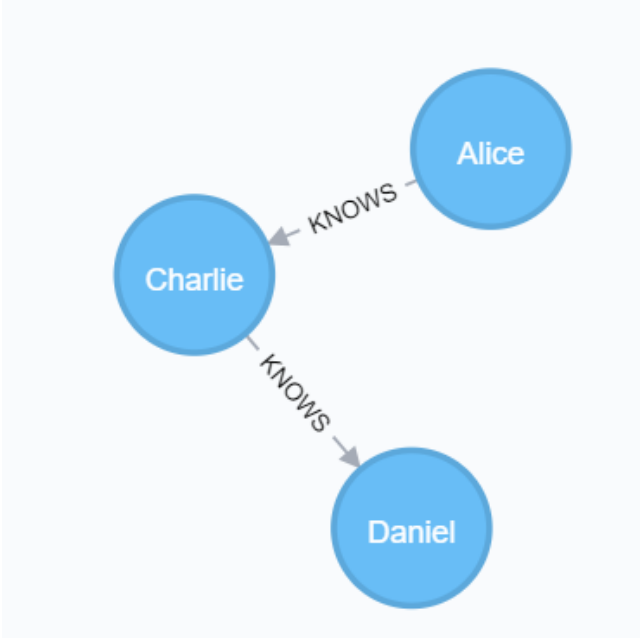


5.4 Neo4j的函数

- 查询一个模式，其中每个节点的age都大于30。

```
MATCH p=(a)-[*2]->(b)
WHERE a.name='Alice' AND b.name='Daniel'
        AND  ALL (x IN nodes(p) WHERE x.age>30)
RETURN p
```





5.4 Neo4j的函数

2）ANY函数

判断一个断言是否至少适用于列表中的其中一个元素。

Any (variable IN list WHERE predicate)

- variable：断言变量
- list：列表表达式
- predicate：测试列表中所有元素的断言

5.4 Neo4j的函数

- 返回所有节点中array数组属性中至少有一个值为‘one’的节点

```
MATCH (a)
WHERE  ANY (x IN a.array WHERE x='one')
RETURN a
```

输出结果：

["one", "two", "three"]

5.4 Neo4j的函数

3）NONE函数

如果断言不适用于列表中的任何元素，则返回true。

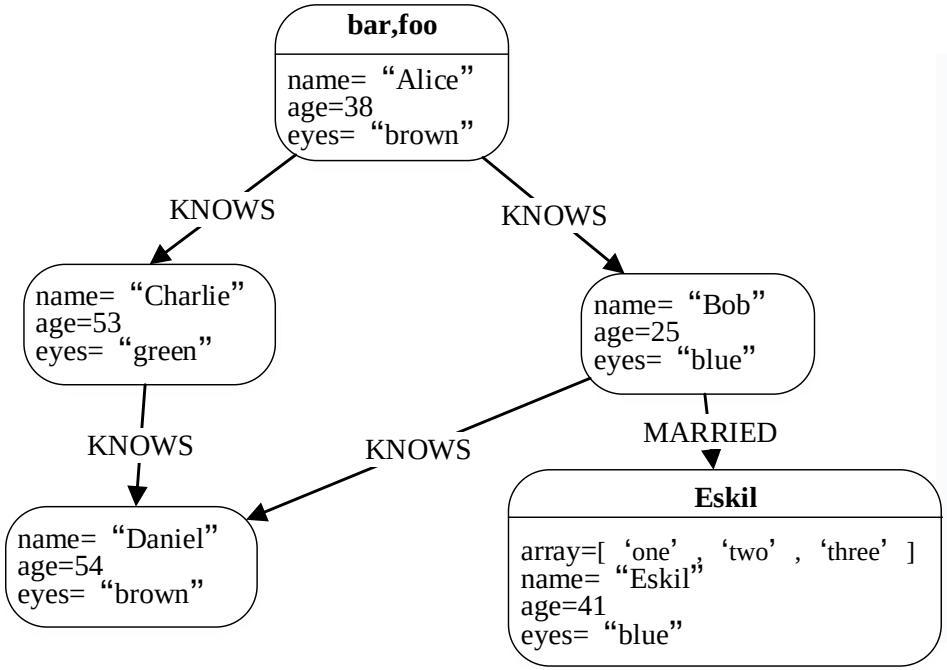
None(variable IN list WHERE predicate)

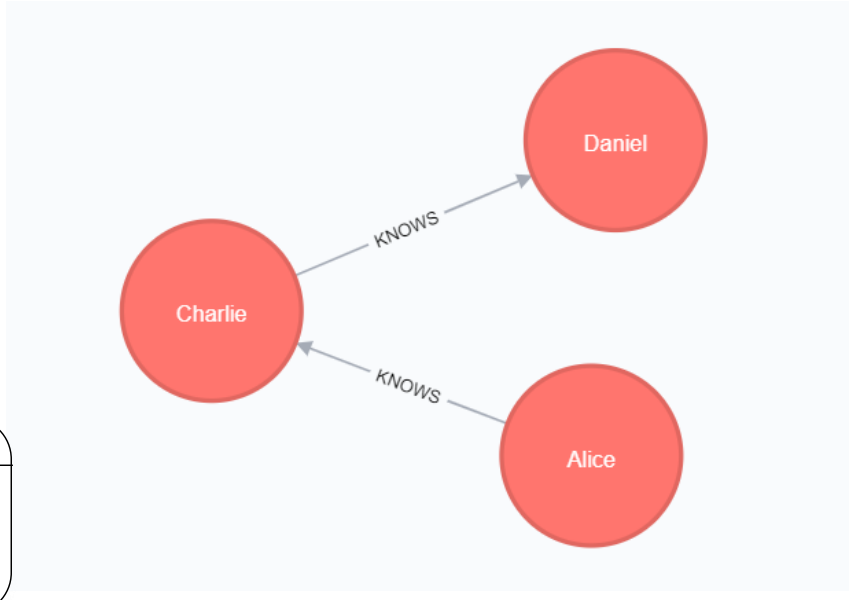
- variable：断言变量
- list：列表表达式
- predicate：测试列表所有元素的断言

5.4 Neo4j的函数

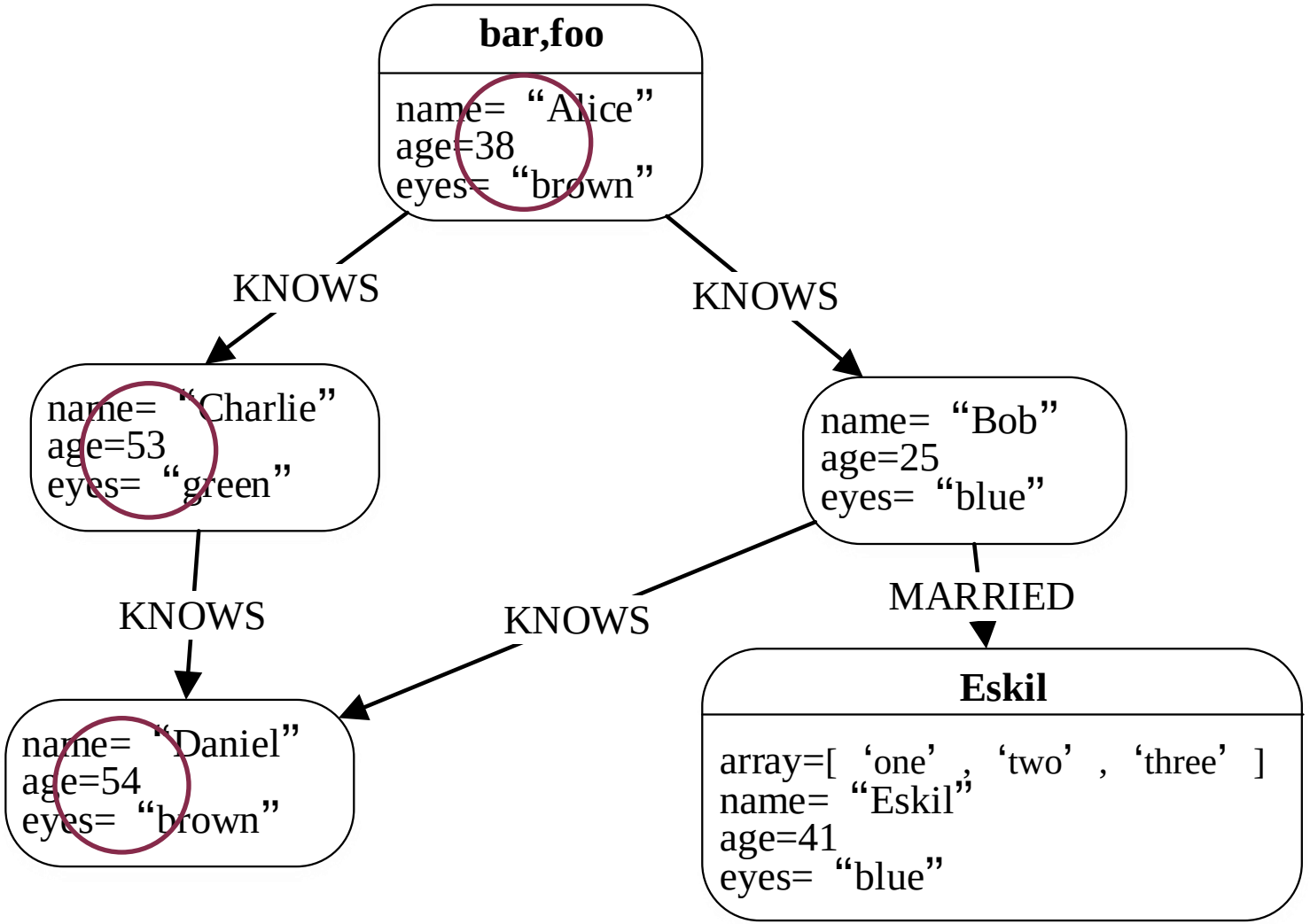
- 返回路径中没有节点的age值为25的路径

```
MATCH p=(n)-[*2]->(b)
WHERE n.name='Alice'
        AND  NONE (x IN nodes(p) WHERE x.age=25)
RETURN p
```





5.4 Neo4j的函数


bar,foo
name= “Alice”
age=38
eyes= “brown”
KNOWS
KNOWS
name= “Charlie”
age=53
eyes= “green”
name= “Bob”
age=25
eyes= “blue”
KNOWS
KNOWS
MARRIED
name= “Daniel”
age=54
eyes= “brown”
Eskil
array=[‘one’ , ‘two’ , ‘three’]
name= “Eskil”
age=41
eyes= “blue”

5.4 Neo4j的函数

4）SINGLE函数

如果断言刚好只适用于列表中的某一个元素，则返回true。

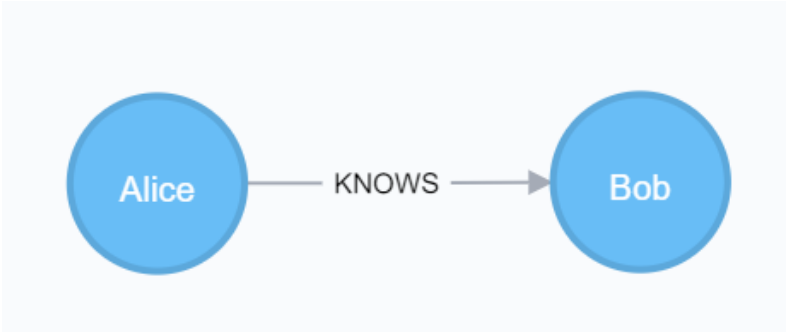
single(variable IN list WHERE predicate)

- variable：断言变量
- list：列表表达式
- predicate：测试列表中所有元素的断言

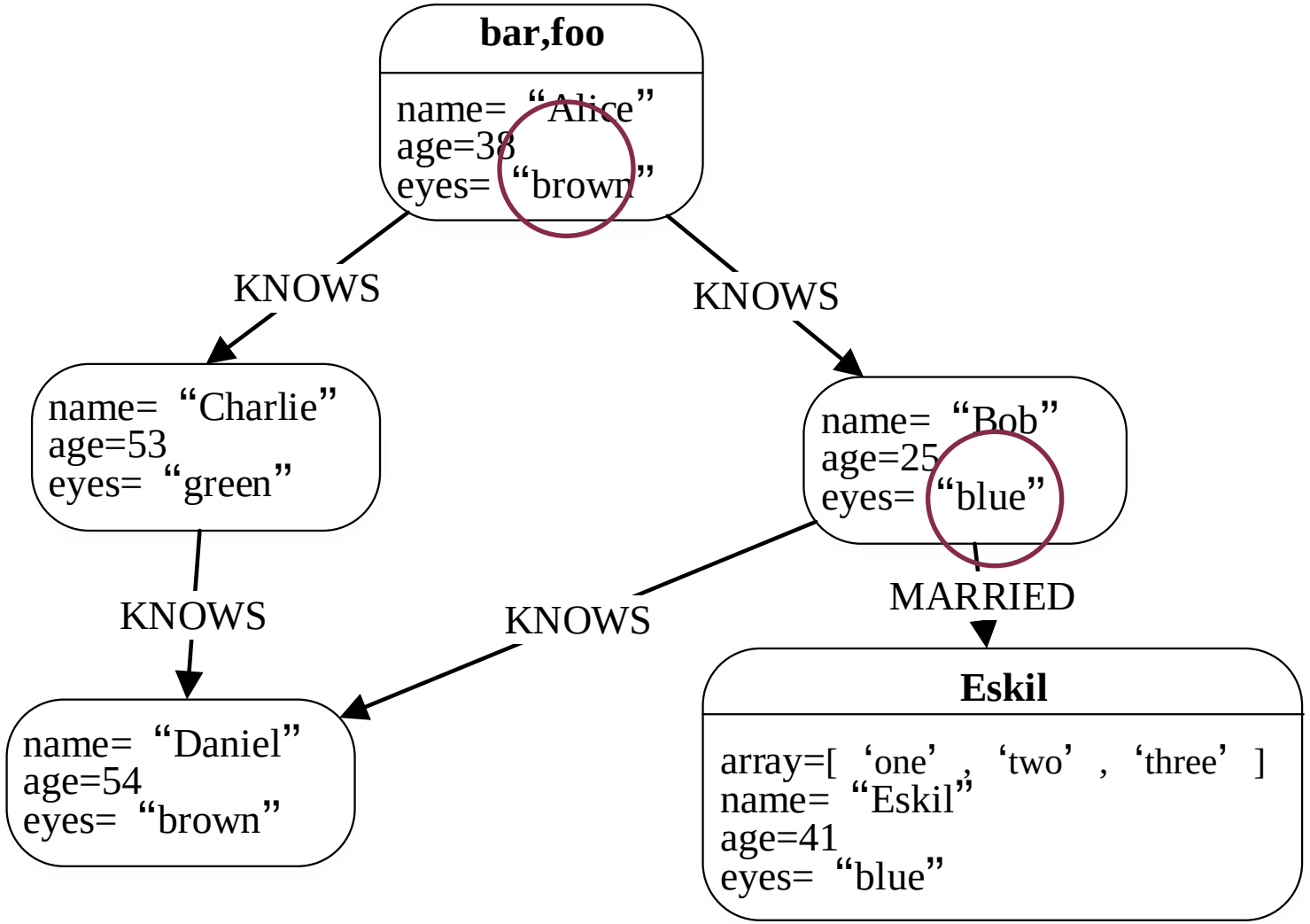
5.4 Neo4j的函数

- 返回路径中刚好只有一个节点的eyes属性值为’blue’的路径。

```
MATCH p=(n)-->(b)
WHERE n.name='Alice'  AND
      SINGLE(var IN nodes(p) WHERE var.eyes='blue')
RETURN p
```



5.4 Neo4j的函数


bar,foo
name= “Alice”
age=38
eyes= “brown”
KNOWS
KNOWS
name= “Charlie”
age=53
eyes= “green”
name= “Bob”
age=25
eyes= “blue”
KNOWS
KNOWS
MARRIED
name= “Daniel”
age=54
eyes= “brown”
Eskil
array=[‘one’ , ‘two’ , ‘three’]
name= “Eskil”
age=41
eyes= “blue”

5.4 Neo4j的函数

5）EXISTS函数

如果数据库中存在指定的模式，或节点中存在指定的属性时，则返回true。

Exists(pattern-or-property)

– pattern-or-property：模式或者属性

5.4 Neo4j的函数

- 返回具有name属性，且是否已婚的信息。

```
MATCH (n)
WHERE exists(n.name) //存在指定name属性
RETURN n.name AS name,
       exists((n)-[:MARRIED]->()) AS is_married
```

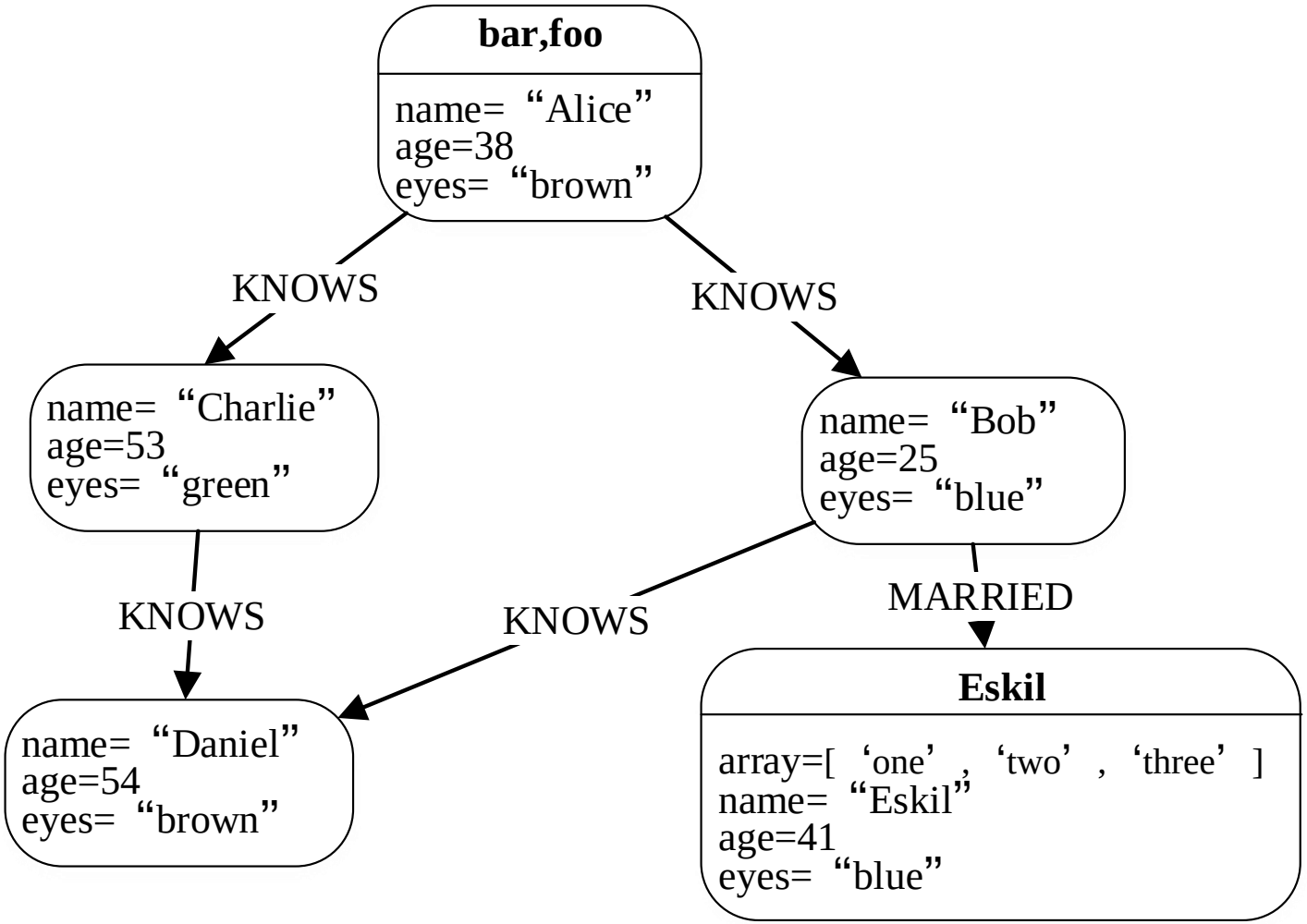
name	is_married
"Alice"	false
"Bob"	true
"Charlie"	false
"Daniel"	false
"Eskil"	false

5.4 Neo4j的函数

2、标量函数：返回一个值

- Size：返回列表中元素的个数。
- Length：返回路径的长度（关系的个数）。
- Type：返回关系的类型(关系的类型只有一个)。
- Id：返回关系或者节点的id.
- Coalesce：返回表达式列表中第一个非空的值，若全为空，返回null。
- Head：返回列表中的第一个元素。
- Last ：返回列表中最后一个元素。
- startNode:返回一个关系的开始节点。
- endNode：返回一个关系的结束节点。

5.4 Neo4j的函数


bar,foo
name= “Alice”
age=38
eyes= “brown”
KNOWS
KNOWS
name= “Charlie”
age=53
eyes= “green”
name= “Bob”
age=25
eyes= “blue”
KNOWS
KNOWS
MARRIED
name= “Daniel”
age=54
eyes= “brown”
Eskil
array=[‘one’ , ‘two’ , ‘three’]
name= “Eskil”
age=41
eyes= “blue”

5.4 Neo4j的函数

1）返回列表中元素的个数

RETURN size(['Alice','Bob']) AS col

2）返回模式表达式匹配的子图的个数

MATCH(a)

WHERE a.name='Alice'

RETURN size((a)-->()-->()) AS num

3）返回路径p的长度

MATCH p=(a)-->(b)-->(c) WHERE a.name='Alice'

RETURN length(p)

4）返回关系的类型

MATCH (n)-[r]-() WHERE n.name='Alice'

RETURN type(r)

5.4 Neo4j的函数

5）返回节点的id

MATCH (a) RETURN id(a)

6）返回列表中第一个非空的属性值

MATCH (a)

WHERE a.name=‘Alice’

RETURN coalesce(a.hairColor, a.eyes) // hairColor为null

7）返回关系的开始节点和结束节点

MATCH (a:Person{name:'Alice' })-[r]-()

RETURN startNode(r), endNode(r)

8）返回节点或关系的属性及属性值

MATCH (b:Person{name:'Bob’})

RETURN properties(b)//返回值是一个集合{第一个属性和值，第二属性和值，…}

5.4 Neo4j的函数

3、列表函数：返回多个值

- Nodes：返回路径中的所有节点的属性和属性值。
- Relationships：返回路径中的所有关系的属性和属性值。
- Labels：返回节点的标签(标签可以有多个)。
- Keys：以字符串形式返回一个节点、关系或map的所有属性名称。
- Tail：返回除首元素之外的所有元素。
- Range：返回某个范围内的数值，值之间默认步长为1。
- Reduce：对列表中每个元素执行表达式，将表达式结果存入累加器。

5.4 Neo4j的函数

1）返回路径中的所有节点(包括节点的属性和属性值)。

```
MATCH p=(a)-->(b)-->(c)
WHERE a.name='Alice' AND c.name='Eskil'
RETURN nodes(p)//返回一个节点列表
```

2）返回路径p中的所有关系（类型及其属性）。

```
MATCH p=(a)-->(b)-->(c)
WHERE a.name='Alice' AND c.name='Eskil’
RETURN relationships(p) //返回一个关系列表
```

3）返回节点a的所有标签。

```
MATCH (a)
WHERE a.name='Alice'
RETURN labels(a)
```

5.4 Neo4j的函数

4）返回节点a的所有属性名。

```
MATCH (a) WHERE a.name='Alice’
RETURN keys(a)
```

5）返回array属性中除了第一个元素之外的其他所有元素。

```
MATCH (a) WHERE a.name= 'Eskil'
RETURN a.array, tail(a.array)
```

6）返回0-10之间步长为1，2-18步长为3的所有值。

```
RETURN range(0,10),range(2,18,3)
```

5.4 Neo4j的函数

- REDUCE函数

对列表中每个元素执行一个表达式，将表达式结果存入累加器

reduce(accumulator=initial,variable IN list|expression)

- 将路径中每个节点的age数值加起来，返回一个单值。

```
MATCH p=(a)-->(b)-->(c)
WHERE a.name='Alice' AND b.name='Bob' And c.name='Daniel'
RETURN reduce(totalAge=0, n IN nodes(p) | totalAge+n.age) AS reduction
```

reduction
96

5.4 Neo4j的函数

4）数学函数

- Abs(expression):返回绝对值
- Round(expression)：返回距离表达式值最近的整数
- Sqrt(expression)： 返回数值的平方根
- Sign(expression)：返回一个数值的正负，正1、负-1、零0
- Floor：返回小于等于表达式的最大整数
- ……

5.4 Neo4j的函数

5）字符串函数

- Replace：返回被替换字符串替换后的字符串。
- Substring：返回原字符串的字串。
- left：返回原字符串左边指定长度的子串。
- right：返回原字符串右边指定长度的子串。
- ltrim：返回原字符串移除左侧的空白字符后的字符串。
- rtrim：返回原字符串移除右侧的空白字符后的字符串。
- trim ：返回原字符串移除两侧的空白字符后的字符串。
- ……

5.5 Neo4j的模式操作

1、模式的概念

模式用于描述如何搜索图数据，包括以下几种：

- 节点模式
- 关系模式
- 关联节点模式
- 变长路径的模式
- 路径变量

5.5 Neo4j的模式操作

1）节点模式

节点模式使用()标识，节点具有标签和属性，如果需要引用节点，则需要给节点变量名：

- ()：该模式用于描述节点，且是匿名节点；
- (n)：该模式用于描述节点，节点的变量名是n;
- (n:label)：该模式用于描述节点，节点具有特定的标签label；也可以指定多个标签；
- (n{name:"Vic"})：该模式用于描述节点，节点具有name属性，并且name属性值是“Vic”；也可以指定多个属性；
- (n:label{name:"Vic"})：该模式用于描述节点，节点具有特定的标签和name属性，并且name属性值是“Vic”；

5.5 Neo4j的模式操作

2) 关系模式

关系模式使用[]标识，在属性图中，节点之间存在关系，节点之间的关系通过箭头()-[]->()表示，例如：

- []：该模式用于描述关系，且是匿名关系。
- [r]：该模式用于描述关系，关系的变量名是r；
- [r: type]：该模式用于描述关系，type是关系类型；每一个关系必须有且仅有一个类型；
- [r:type{name:"Friend"}]：该模式用于描述关系，关系的类型是type，关系具有属性name，并且name属性值是“Friend”；

5.5 Neo4j的模式操作

3) 关联节点模式

节点之间通过关系联系在一起，由于关系具有方向性，因此，-->表示存在有向关系，--表示存在关联关系，不指定关系的方向，例如：

- (a)-[r]->(b)：该模式用于描述节点a和b之间存在有向的关系r，
- (a)-->(b)：该模式用于描述a和b之间存在有向关系；

5.5 Neo4j的模式操作

4) 变长路径的模式

从一个节点通过关系连接到另外一个节点，这个过程叫遍历，经过的节点和关系的组合叫做路径（Path），路径是节点和关系的有序组合。

- (a)-->(b)：是步长为1的路径，节点a和b之间有关系直接关联；
- (a)-->()-->(b)：是步长为2的路径，从节点a，经过两个关系和一个节点，到达节点b；

5.5 Neo4j的模式操作

Cypher语言支持变长路径的模式，变长路径的表示方式是：[*N..M]，N和M表示路径长度的最小值和最大值。

- (a)-[*2]->(b)：表示路径长度固定为2，起始节点是a，终止节点是b；
- (a)-[*3..5]->(b)：表示路径长度的最小值是3，最大值是5，起始节点是a，终止节点是b；
- (a)-[*3..]->(b)：表示路径长度的最小值是3，起始节点是a，终止节点是b；
- (a)-[*..5]->(b)：表示路径长度的最大值是5，起始节点是a，终止节点是b；
- (a)-[*]->(b)：表示不限制路径长度，起始节点是a，终止节点是b；

5.5 Neo4j的模式操作

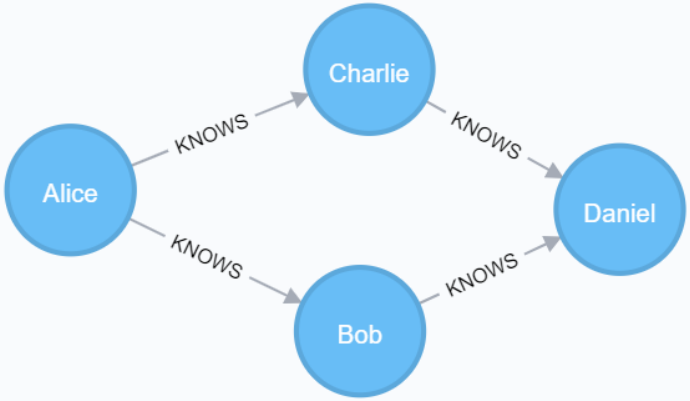
5) 路径变量

路径可以指定（assign）给一个变量，该变量是路径变量，用于引用查询路径。

– p = (a)-[*3..5]->(b)

- **返回‘Alice‘节点到’Daniel‘节点，且长度不超过3的路径。**

```
MATCH p=(a)-[*1..3]->(b)
WHERE a.name='Alice' AND
b.name='Daniel'
RETURN nodes(p)
```



5.5 Neo4j的模式操作

2、索引

Cypher允许节点的某个属性上有特定的索引。索引一旦创建，它将自己管理并当图发生变化时自动更新。一旦索引创建并生效之后，就自动使用索引。

5.5 Neo4j的模式操作

1）创建索引

使用CREATE INDEX可以在拥有某个标签的所有节点的某个属性上创建索引。

- 在拥有Person标签的节点name属性上创建索引

CREATE INDEX ON :Person(name)

标签类似关系数据库中的表名，属性名就相当于表的列名。这里可以看做是在Person表的name列上建立了索引。

5.5 Neo4j的模式操作

2）删除索引

使用DROP INDEX可以删除拥有某个标签的所有节点的某个属性上的索引。

- 在删除Person标签的节点name属性上的索引

DROP INDEX ON :Person(name)

No data returned.
Indexes removed :1

5.5 Neo4j的模式操作

3、约束

Neo4j通过使用约束来保证数据完整性。约束可应用于节点或者关系，可以创建节点属性的唯一性约束，也可以创建节点和关系的属性存在性约束。

- 属性的**唯一性约束**：确保属性值是唯一的，这个规则不适用于没有该属性的节点。
- 属性的**存在性约束**：确保属性是存在的。试图创建没有该属性的节点或关系，以及试图删除强制属性的查询都将失败。
- 唯一性约束和存在性约束可以同时添加到同一个属性上。

只有Neo4j企业版才具有属性存在性约束这个高级功能。

5.5 Neo4j的模式操作

1）创建节点属性的唯一性约束

使用IS UNIQUE语法创建约束，它能确保数据库中拥有特定标签和属性值的节点是唯一的。

- 给Person标签的所有节点的name属性添加唯一性约束

```
CREATE  CONSTRAINT ON (b:Person)
ASSERT b.name IS UNIQUE
```

5.5 Neo4j的模式操作

2）删除节点属性的唯一性约束

使用DROP CONSTRAINT删除数据库中的一个约束。

- 删除Person标签节点name属性的唯一性约束

DROP CONSTRAINT ON(b:Person)

ASSERT b.name IS UNIQUE

5.5 Neo4j的模式操作

3）创建节点属性的存在性约束

创建存在性约束可确保有标签的节点都有一个特定的属性。

- 给具有Person标签的节点name属性添加存在性约束

```
CREATE CONSTRAINT ON(p:Person)
ASSERT exists(p.name)
```

5.5 Neo4j的模式操作

4）删除节点属性的存在性约束

使用DROP CONSTARINT 可以从数据库中移除一个约束。

- 删除Person标签下所有节点的name属性的存在性约束

DROP CONSTRAINT ON (p:Person)

ASSERT exists(p.name)

5.5 Neo4j的模式操作

5）创建关系属性的存在性约束

创建存在性约束可确保特定类型的所有关系都有一个特定的属性。

- 给Knows类型下所有关系的属性day添加存在性约束

CREATA CONSTRAINT ON ()-[k: Knows]-()

ASSERT exists(k.day)

5.5 Neo4j的模式操作

6）删除关系属性的存在性约束

使用DROP CONSTARINT 可以从数据库中移除一个关系属性的存在性约束。

- 删除Knows类型下所有关系的day属性的存在性约束

DROP CONSTRAINT ON ()-[k: Knows]-()

ASSERT exists(k.day)

5.6 应用案例

将关系型数据库用图数据库存储

Students基本表

Sno	Sname	Ssex	Sage	Dno
S01	王建平	男	21	D01
S02	刘华	女	19	D01
S03	范林军	女	18	D02
S04	李伟	男	19	D03
S05	黄河	男	18	D03
S06	长江	男	20	D03

5.6 应用案例

Reports基本表

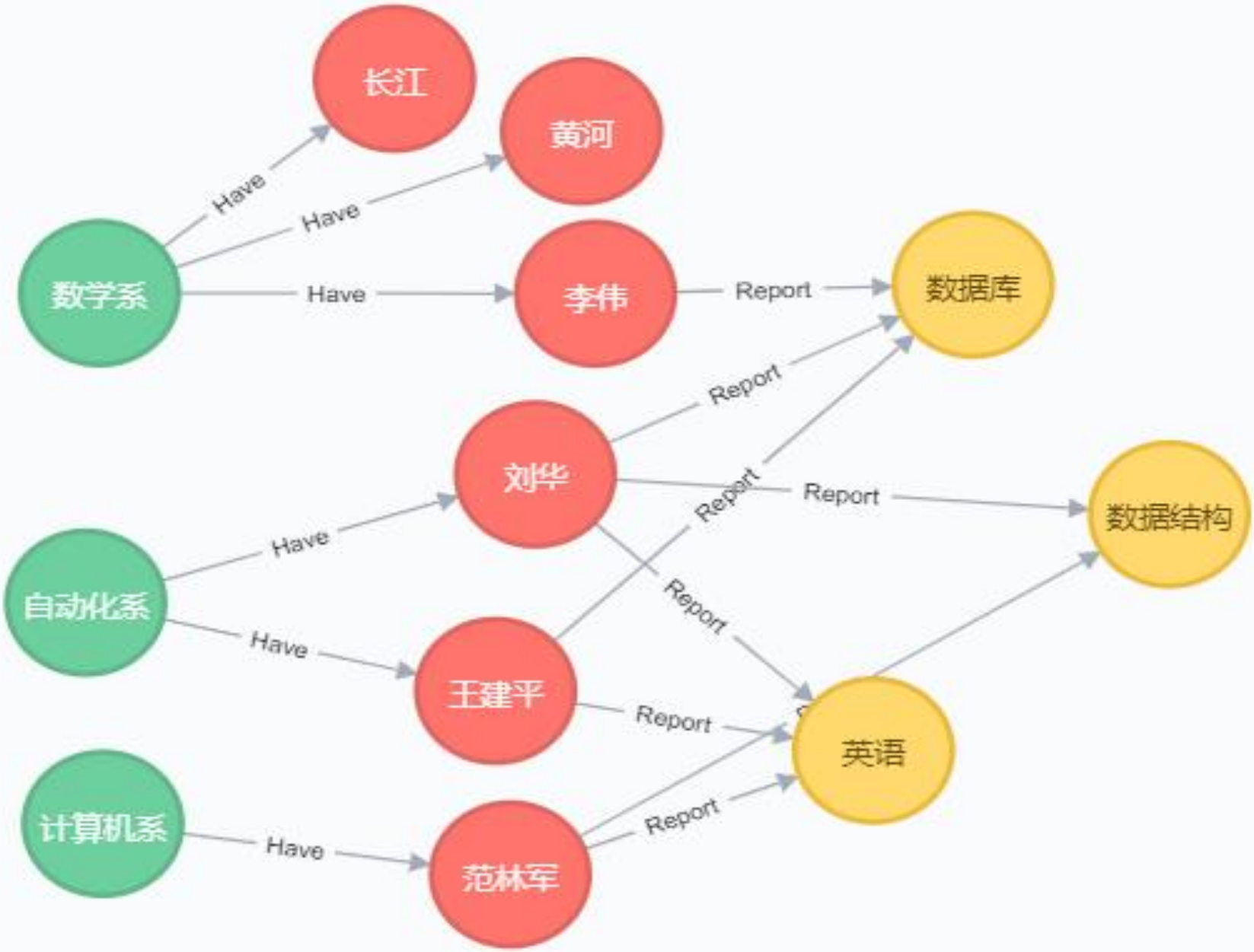
Sno	Cno	Grade
S01	C01	92
S01	C03	84
S02	C01	90
S02	C02	94
S02	C03	82
S03	C01	72
S03	C02	90
S04	C03	75

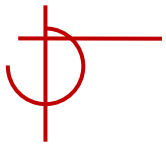
Courses基本表

Cno	Cname	Pre_Cno	Credits
C01	英语		4
C02	数据结构	C01	2
C03	数据库	C02	2

Depts基本表

Dno	Dname
D01	自动化系
D02	计算机系
D03	数学系


长江
黄河
数学系
Have
Have
Have
李伟
Report
数据库
Report
刘华
Report
Report
数据结构
Report
自动化系
Have
Have
王建平
Report
英语
计算机系
Have
范林军
Report



Chapter over