语义网与知识图谱

上海大学计算机学院

主讲： 刘 炜

《语义网与知识图谱》

知识存储

上海大学计算机学院 刘炜

2020年10月

内容概览

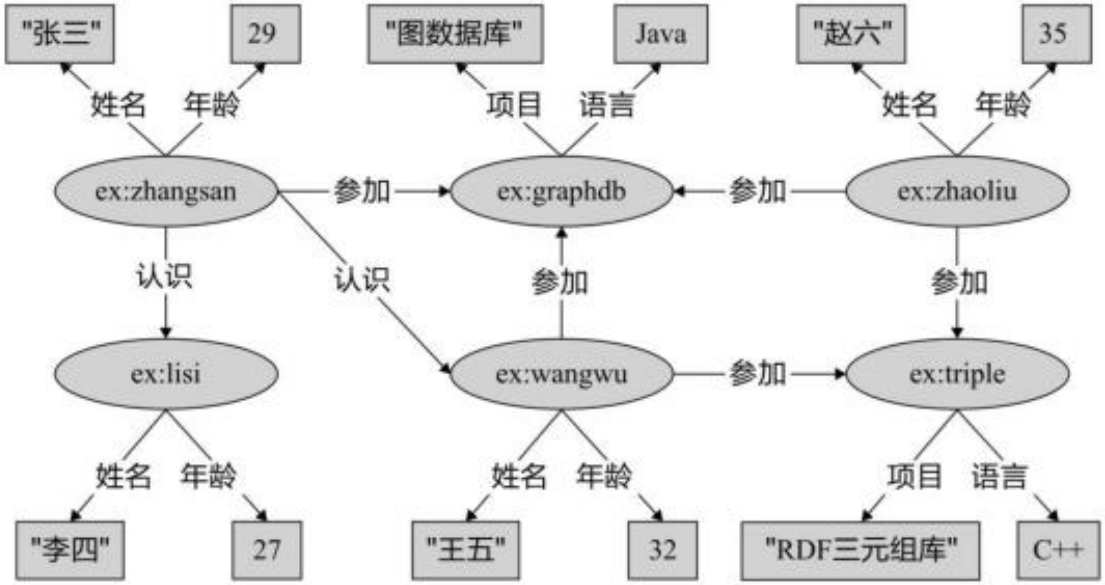
一、知识图谱存储

1. 前言
2. 知识图谱数据库基础知识
3. 常见知识图谱存储方法
4. 原生图数据库存储原理

知识图谱存储

- 知识图谱存储面临的问题：
  - 以文件形式保存知识图谱无法满足用户的查询、检索、推理、分析等各种应用需求；
  - 传统的关系模型与知识图谱的图模型之间存在显著差异，关系数据库无法有效地管理大规模的知识图谱数据。
- 语义Web领域发展出**专门存储RDF数据的三元组库**；
- 数据库领域发展出用于管理属性图的**图数据库**，如Neo4j；
- 知识图谱的存储需要综合考虑图的特点、复杂的知识结构存储、索引和查询的优化等问题。
- 典型的知识存储引擎分为**基于关系数据库的存储、面向RDF的三元组数据库和基于原生图的存储。**
- 在实践中，对知识的存储多为混合存储结构，图数据库存储并非必须，例如Wikidata项目目后端是MySQL实现的。

知识图谱数据模型——RDF图



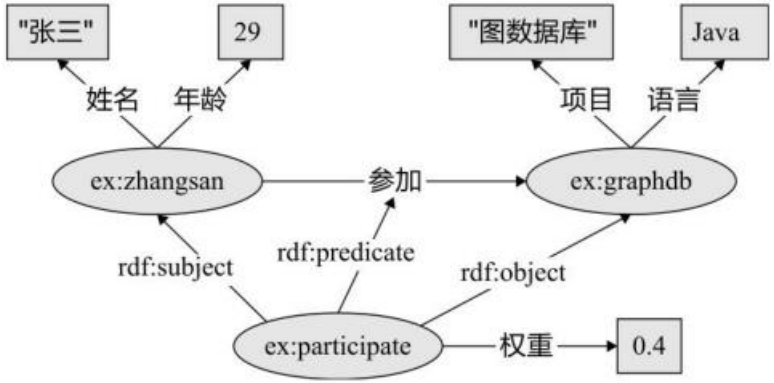
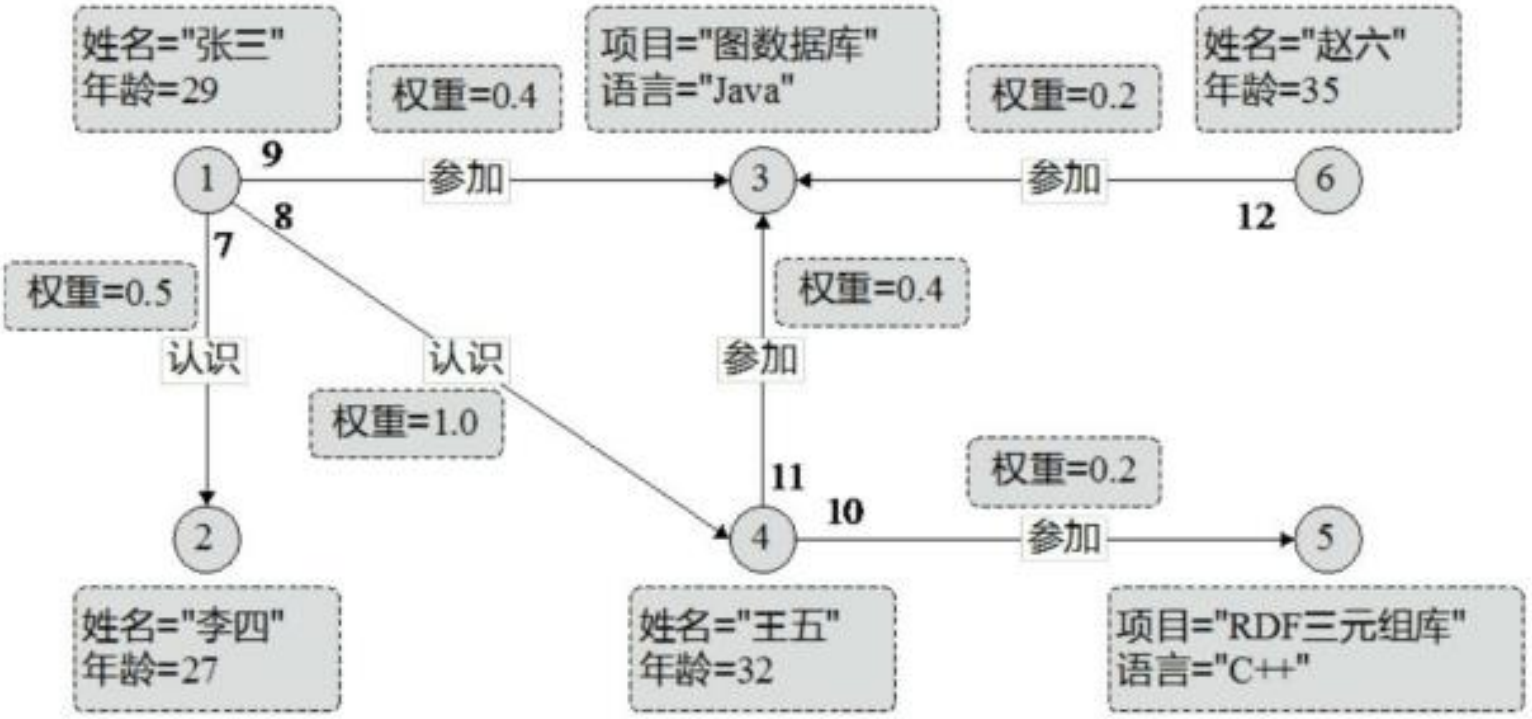


图3-2 RDF图中边属性的表示

知识图谱数据模型——属性图

- 属性图是目前被图数据库业界采纳最广的一种图数据模型。
- 属性图由节点和边集组成，且满足以下特性：
  - ✓ 每个节点具有唯一的ID;
  - ✓ 每个节点具有若干条出边；
  - ✓ 每个节点具有若干条入边；
  - ✓ 每个节点具有一组属性，每个属性是一个键值对；
  - ✓ 每条边具有 唯一 的id;
  - ✓ 每条边具有一个头节点；
  - ✓ 每条边具有一个尾节点；
  - ✓ 每条边具有一个标签表示联系；
  - ✓ 每条边具有一组属性，每个属性是一个键值对；

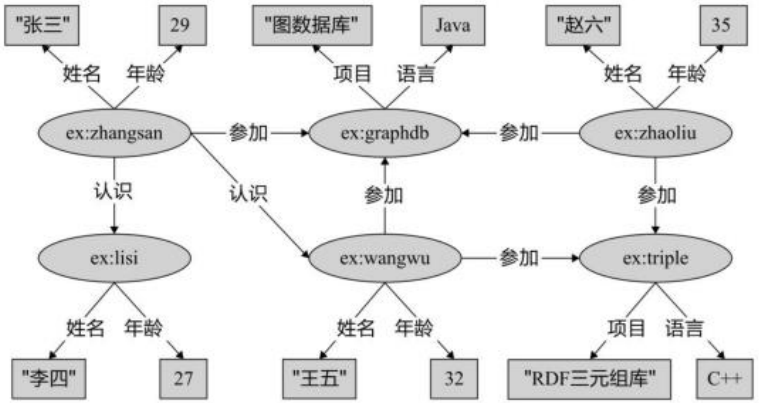
知识图谱数据模型——属性图



知识图谱查询语言

- 包括：RDF图上的查询语言SPARQL，属性图上的常用的查询语言Cypher和Gremlin.
- SPARQL 1.1:
  - ✓ W3C制定的RDF图数据标准查询语言；
  - ✓ 借鉴了SQL，属于声明式查询语言；
  - ✓ 设计了三元组模式、子图模式、属性路径等多种查询机制；
  - ✓ 几乎所有的RDF三元组数据库都实现了SPARQL。

SPARQL查询实例



- **实例1：查询程序员张三认识的其他程序员**

```
PREFIX ex: <http://www.example.com />
SELECT ?p
WHERE {   ex:zhangsan  ex:knows  ?p  .  }

输出：ex: lisi
      ex:wangwu
```

- **实例2：查询程序员张三认识的其他程序员参加的项目**

```
PREFIX ex: <http://www.example.com />
SELECT ?pr
WHERE {   ex:zhangsan  ex:knows  ?p  .
            ?p   ex:participat ?pr . }

输出：ex: graphdb
      ex: triple
```

两个三元组模式组成的基本图模式（BGP）

查询参加“图数据库”项目的所有程序员。

SPARQL查询实例

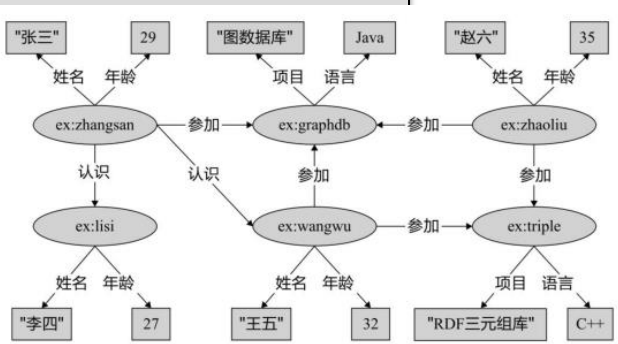
- **实例3：查询程序员张三认识的30岁以上的程序员参加的项目名称**

```
PREFIX ex: <http://www.example.com />
SELECT ?name
WHERE {   ex:zhangsan  ex:knows  ?p  .
          ?p ex:age ?age.
          FILTER(?age>30)
          ?p ex:participate ?pr.
          ?pr rdfs:label  ?name .}
```

输出：图数据库
RDF三元组库

链式+星型结构

查询过程是在数据图中寻找与查询图映射匹配的所有子图。



- **实例4：查询年龄为29的参加了ex:graphdb项目的程序员参加的其他项目及其直接或间接认识的程**

查询参加“ex:triple”项目的所有年龄小于35的程序员参加的其他项目。

```
PREFIX ex: <http://www.example.com />
SELECT ?name
WHERE {   ?p  ex:participate  ex:graphdb  .
          ?p ex:age 29.
          ?p ex:knows */ex:participate ?pr.
          ?pr rdfs:label  ?name .}
```

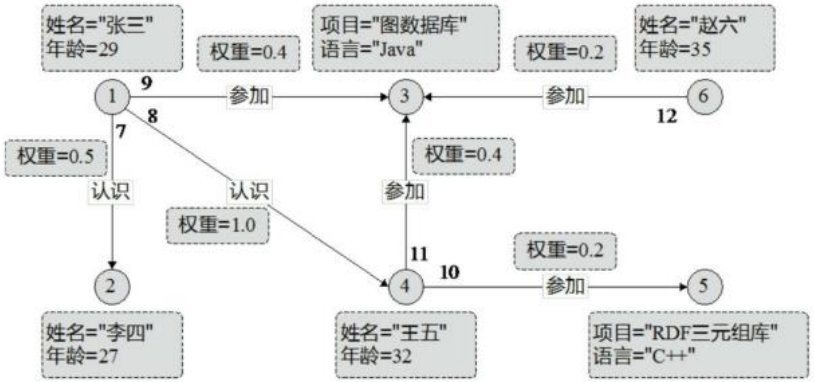
输出：图数据库
RDF三元组库

属性路径机制，ex:knows*/ex:participate类似于正则表达式，表达经过0条，1条或多条ex:knows边，再经过一条ex:participate边。

知识图谱查询语言——Cypher

- Cypher： 最初是图数据库Neo4j实现的属性图数据查询语言。
  - 声明式查询语言；
  - SAP HANA Graph、Redis Graph、AgensGraph和Memgraph等图数据库都已实现了Cypher。

Cypher语法手册：
https://neo4j.com/docs/cypher-manual/current/



- 实例1：查询图中所有的程序员节点

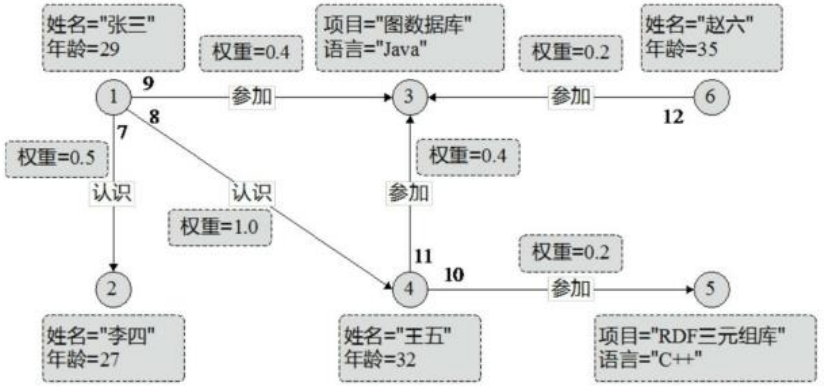
```
MATCH (p:程序员)
RETURN p
输出：
    {姓名=张三，年龄=29}
    {姓名=李四，年龄=27}
    {姓名=王五，年龄=32}
    {姓名=赵六，年龄=35}
```

查询图中所有的项目节点。

知识图谱查询语言——Cypher

- **实例2：查询程序员与“图数据库”项目之间的边**

```
MATCH (:程序员) – [ r ]-> (:项目{name: ‘图数据库’ })
RETURN r
输出：
    (1) – [参加{权重=0.4}] -> (3)
    (4) – [参加{权重=0.4}] -> (3)
    (6) – [参加{权重=0.2}] -> (3)
```



返回边及其属性，程序员类别的节点

- **实例3：查询从节点1出发的标签为“认识”的边**

```
MATCH (1:程序员) – [ r:认识 ]-> ()
RETURN r
输出：
    (1) – [认识{权重=0.5}] -> (2)
    (1) – [认识{权重=1.0}] -> (4)
```

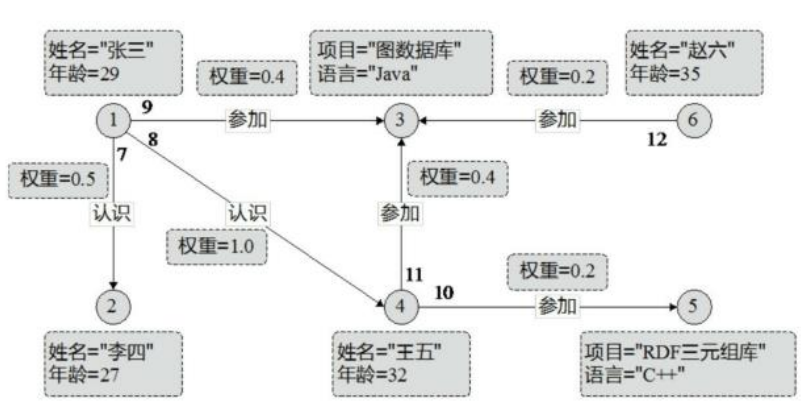
查询图中“王五”参加的所有项目名字。

知识图谱查询语言——Cypher

- **实例4：查询节点1认识的30岁以上的程序员参加的项目**

```
MATCH (1:程序员) – [ :认识 ]-> (p:程序员), (p)-[:参加]->(pr:项目)
WHERE p.年龄>30
RETURN pr.项目
输出：
    图数据库
    RDF三元组库
```

等价与SPARQL BGP的链式查询。



查询图中参加节点3的年龄小于30岁的程序员认识的其他程序员。

- **实例5：查询年龄为29的参加了项目3的程序员参加的其他项目及其直接或间接认识的程序员参加的项目**

```
MATCH (p:程序员{年龄:29}) – [:参加 ]-> (3:项目), (p)-[:认识*0..]->()-[:参加]->(pr:项目)
RETURN pr
输出：
    {项目=图数据库，语言=java}
    {项目=RDF三元组库，语言=C++}
```

“认识*0..”表示一个节点达到另一个节点的路径包括0个、1个或多个“认识”边。

知识图谱存储方法——基于关系数据库的方案

- 基于关系数据库的存储方案是目前知识图谱采用的一种主要的存储方法。
- 知识图谱主要的存储结构：
  - 三元组表
  - 水平表
  - 属性表
  - 垂直划分
  - 六重索引
  - DB2RDF

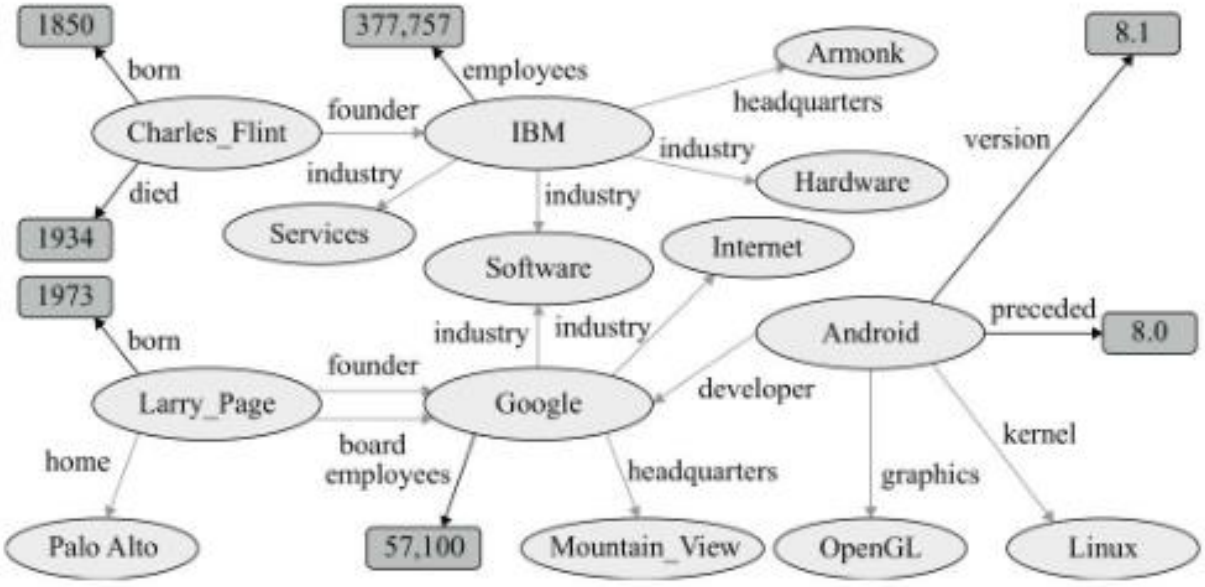
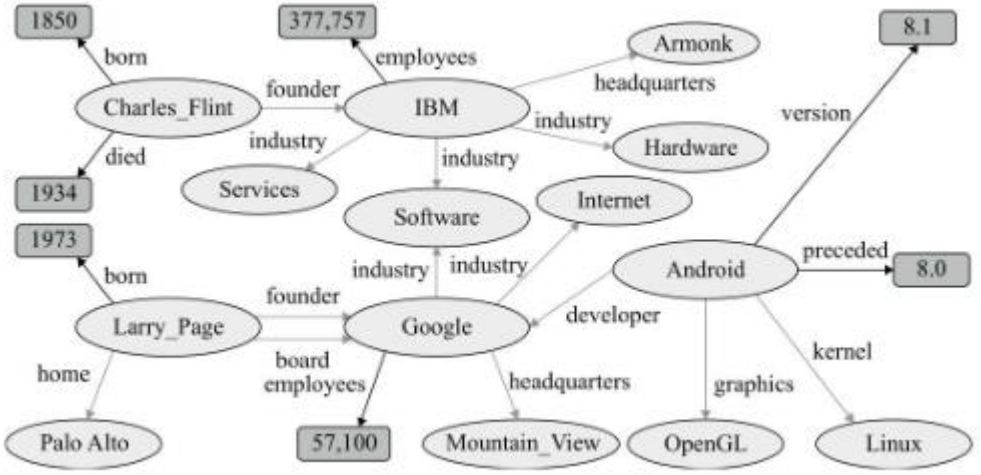


图3-4 摘自DBpedia数据集的RDF知识图谱

基于关系数据库的方案——三元组表

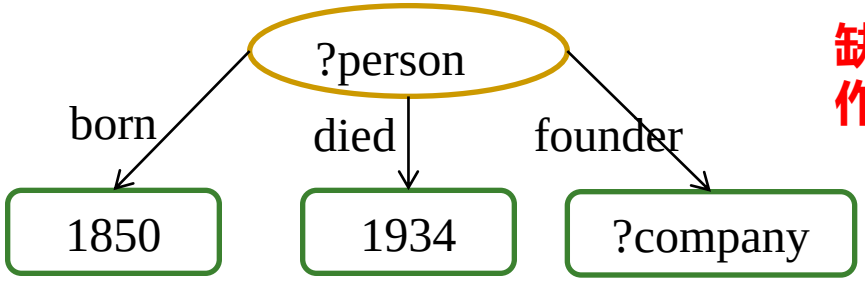
- 三元组表示将知识图谱存储到关系数据库最简单、最直接 的方法。
- 表的模式：三元组表（主语、谓语、宾语）

主语	谓语	宾语
Charles_Flint	Born	1850
Charles_Flint	Died	1934
Charles_Flint	Founder	IBM
Larry_Page	Born	1973
Larry_Page	founder	Google
…		



基于关系数据库的方案——三元组表

主语	谓语	宾语
Charles_Flint	Born	1850
Charles_Flint	Died	1934
Charles_Flint	Founder	IBM
Larry_Page	Born	1973
Larry_Page	founder	Google
…		



缺点：当三元组规模较大时，多个自连接操作会使SQL查询性能低下。

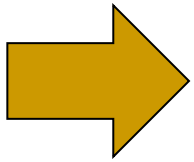
```
SELECT ?person
WHERE {
    ?person born  “1850”  .
    ?person died  “1934”  .
    ?person founder ?company
}
```

```
SELECT t1.主语
FROM t as t1, t as t2, t as t3
WHERE
t1.主语=t2.主语 and t2.主语=t3.主语
and t1.谓语= ‘born’  and t1.宾语= ‘1850’
and t2.谓语= ‘died’  and t2.宾语= ‘1934’
And t3.谓语= ‘founder’
```

基于关系数据库的方案——水平表

主语	Born	died	founder	board	…	employees	headquarters
Charles_Flint	1850	1934	IBM		…		
Larry_Page	1973		Google	Google	…		
Android					…		
Google					…		
IBM					…	54604	Mountain_view
…					…	433,362	Armonk

```
SELECT ?person
WHERE {
    ?person born  “1850”  .
    ?person died  “1934”  .
    ?person founder ?company
}
```



```
SELECT 主语
FROM t
WHERE
born= ‘1850’  and died= ‘1934’
and founder LIKE  ‘_%’
```

优点：查询大大简化，不用进行表连接操作。

缺点：列太多，可能超出上限，空值过多影响表的存储、索引和查询性能。

基于关系数据库的方案——属性表

person

主语	Born	died	founder	board	home
Charles_Flint	1850	1934	IBM		…
Larry_Page	1973		Google	Google	Palo_alto

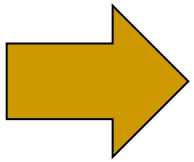
OS

主语	developer	version	kernel	preceded
Android	Google	8.1	Linux	8.0

Company

主语	industry	Employees	headerquarter
IBM	Software, hardware, services	433362	Armonk
Google	Software, internet	54604	Mountain_view

```
SELECT ?person
WHERE {
   ?person born  “1850”  .
   ?person died  “1934”  .
   ?person founder ?company
}
```



```
SELECT 主语
FROM person
WHERE
born= ‘1850’  and died= ‘1934’
and founder LIKE  ‘_%’
```

优点：解决了三元组表自连接的问题和水平表列数过多的问题，缓解空值问题。

缺点：对于规模大的图谱数据，主语的类别多，需要建立成千上万个表，也会超出数据库的限制。对于复杂的查询，仍然要进行多表连接操作，影响效率，也存在空值问题。

基于关系数据库的方案——垂直划分

born

主语	宾语
Charles_Flint	1850
Larry_Page	1973

died

主语	宾语
Charles_Flint	1934

founder

主语	宾语
Charles_Flint	IBM
Larry Page	Google

board

主语	宾语
Larry Page	Google

home

主语	宾语
Larry Page	Palo_Alto

version

主语	宾语
Android	8.1

kernel

主语	宾语
Android	Linux

employees

主语	宾语
IBM	57100
Google	377747

- **把知识图谱划分为若干个只包含主语和宾语的表，表的数量等于谓语的数量。**

```
SELECT ?person
WHERE {
   ?person born  “1850”  .
   ?person died  “1934”  .
   ?person founder ?company
}
```

```
SELECT born.主语
FROM born, died, founder
WHERE
Born.宾语= ‘1850’ and died.宾语= ‘1934’
And born.主语=died.主语 and born.主语
=founder.主语
```

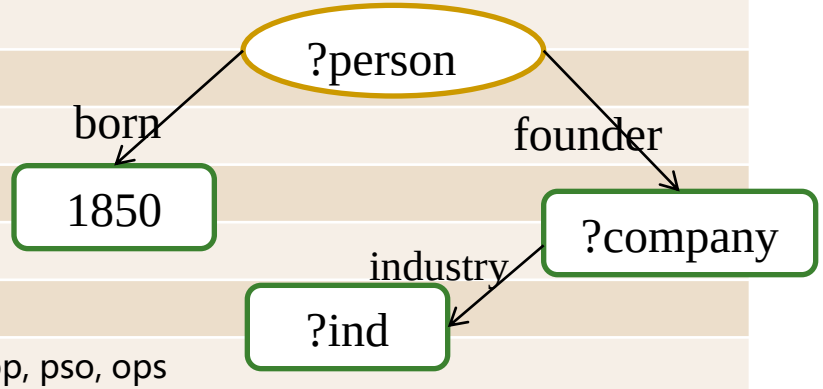
优点：谓语表仅存储出现在知识图谱中的三元组，解决了空值问题；一个主语的一对多联系或多值属性存储在谓语表的多行中，解决了多值问题；能够使用归并排序连接快速执行不同谓语表的链接查询操作。

缺点：大规模图谱，表的数目超过几千个，增加数据开销，越是复杂的查询操作，执行的表连接操作越多，数据更新维护代价大，更新一个主语涉及多张表时，增加I/O开销。

基于关系数据库的方案——六重索引

- **空间换时间，将三元组的六种排列对应地建立6张表，SPO,POS,OSP,SOP,PSO,OPS**

序号	三元组查询模式	可用索引表
1	(s , p, o)	spo, pos, osp, sop, pso, ops
2	(s , p, ?x)	spo, pso
3	(s , ?x, o)	Sop, osp
4	(?x , p, o)	Pos, ops
5	(s , ?x, o)	Spo, sop
6	(s , ?x, ?y)	Osp, ops
7	(?x , p, ?y)	Pos, pso
8	(?x , ?y, ?z)	Spo, pos, osp, sop, pso, ops




```
SELECT ?person, ?ind
WHERE {
   ?person born  “1850”  .
   ?person founder ?company .
   ?company industry ?ind .
}
```

```
SELECT s1.s, p1.o
FROM spo as s1, pso as p1
WHERE
   s1.o=  ‘1850’  and s1.p=  ‘founder’
    and p1.p=  “industry”  and  s1.o=p1.s
```

优点：空间换时间，避免了单表的自连接

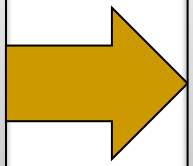
缺点：花费6倍的存储空间，索引维护代价和数据更新代价，随着图谱规模增大，愈加突出。

当图谱查询复杂时，产生大量的连接索引表查询，自连接不可避免。

基于关系数据库的方案—DB2RDF

- **是以往RDF关系存储方案的一种权衡折中，具备了三元组表、属性表和垂直划分方案的部分优点，又克服了部分缺点。**
- **将表的列作为谓语和宾语的存储位置，而不将列与谓语进行绑定。当插入数据时，将谓语动态地映射存储到某列：方案能够确保将相同的谓语映射到同一组列上。**
- **DB2RDF方案由4张表组成，即DPH，RPH，DS和RS表。**

```
SELECT ?person
WHERE {
   ?person born  “1850”  .
   ?person died  “1934”  .
   ?person founder ?company .
}
```

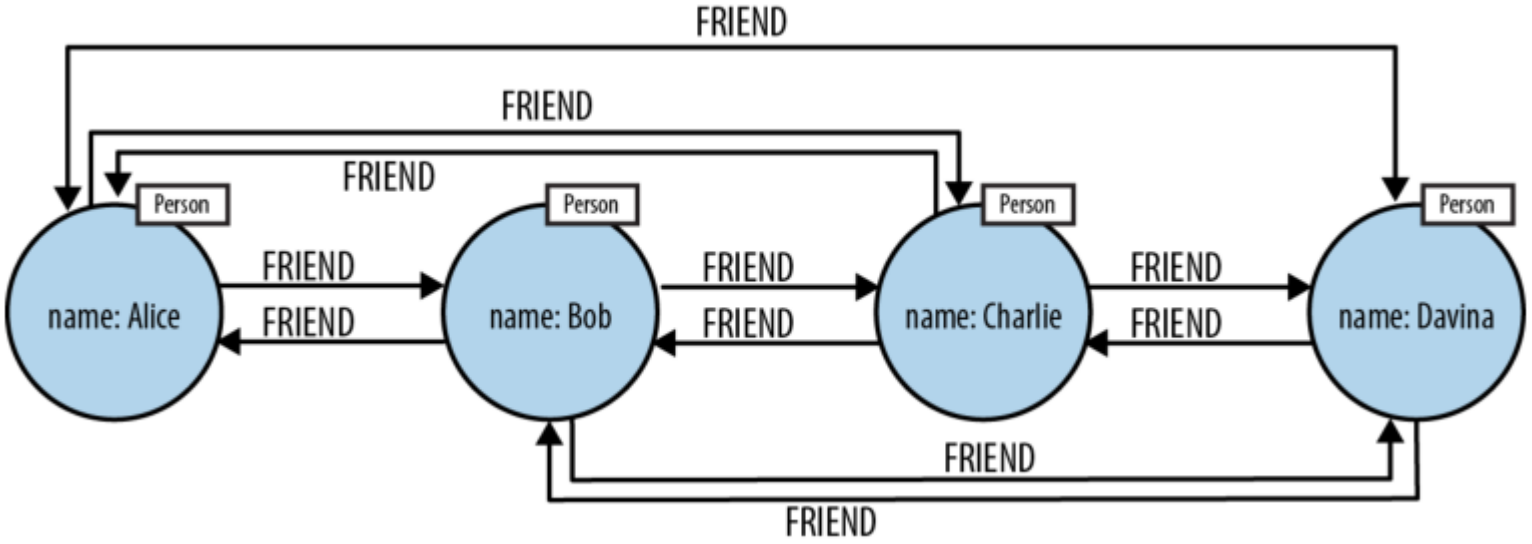


```
SELECT t.主语
FROM dph as t
WHERE
   t.pred1 =  ‘died’  and t.val1 =  ‘1934’
   and t.pred2 =  ‘born’  and
   t.val2=  ‘1850’  and t.pred3 =  ‘founder’
```

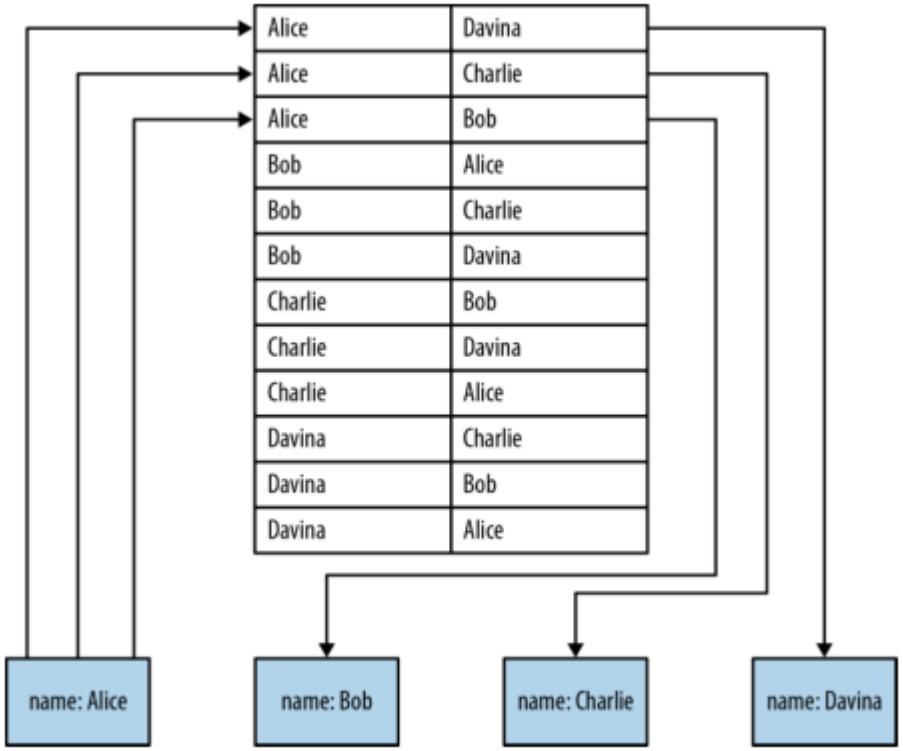
DB2RDF的关键：谓语到列的映射，转化为图着色问题，将一个主语上出现的不同谓语称为共现谓语，目标是让共现谓语着上不同的色（映射到不同的列），非共现谓语可以着上相同的色（映射到同一列）。利用着色冲突图解决。

原生图数据库的实现原理

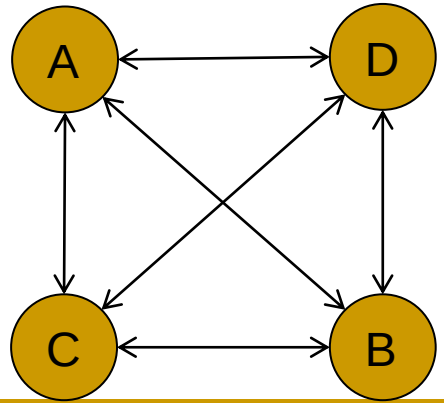
- 原生图是指采用免索引邻接（Index-free adjacency）构建的图数据库引擎，如：AllegroGraph, Neo4j等。
- 采用免索引邻接的数据库为每一个节点维护了一组指向其相邻节点的引用，这组引用本质上可以看做是相邻节点的微索引（Micro Index）或局部索引。
- 这种微索引比起全局索引在处理图遍历查询时非常廉价，其查询复杂度与数据集整体大小无关，仅正比于相邻子图的大小。



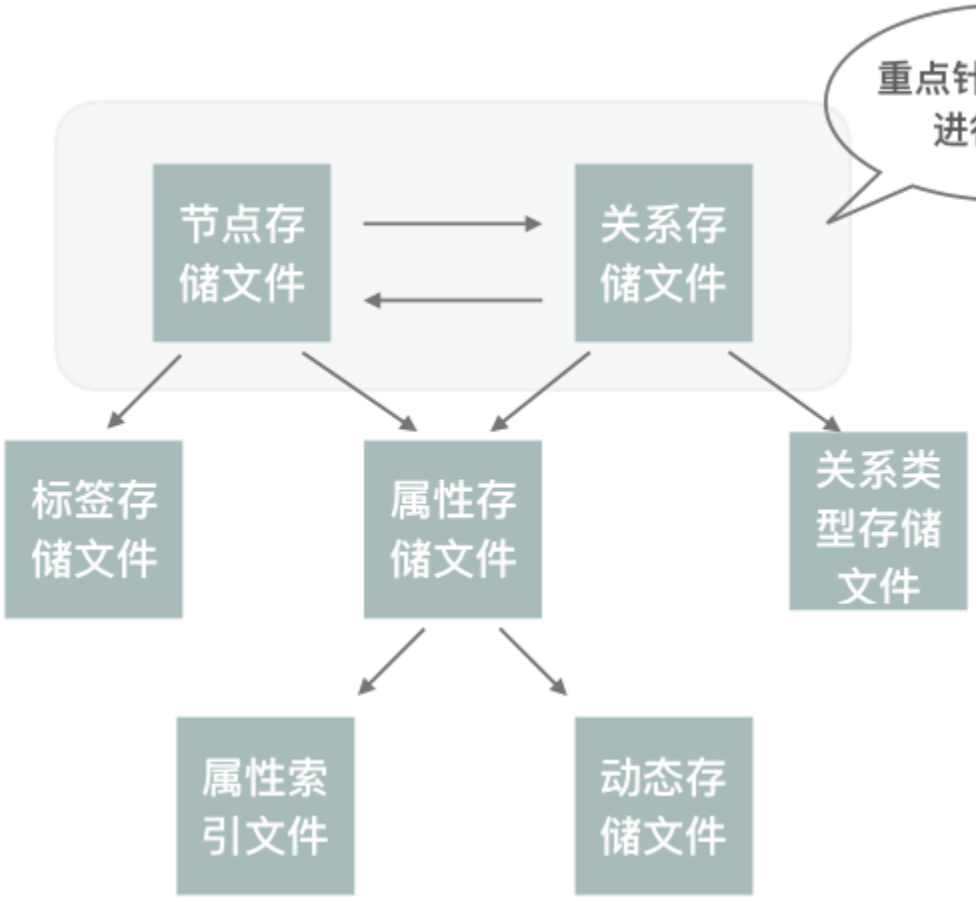
非原生图与原生图的计算复杂度比较



1. 采用全局索引查找，找到Alice的朋友代价为O(log n)
2. 为了找到谁与Alice做朋友，我们需要执行多次全局索引，因为有1个或者每个人都和Alice做朋友.
3. 因此找到谁是Alice的朋友代价是 O(log n)，找到谁与Alice做朋友的代价是 O(m log n)。
4. 采用免索引邻接，可以从节点的两个方向开始遍历边，找到Alice的朋友，只要从Alice节点的出边导航，操作的代价为O(1).



原生图数据库的物理存储实现



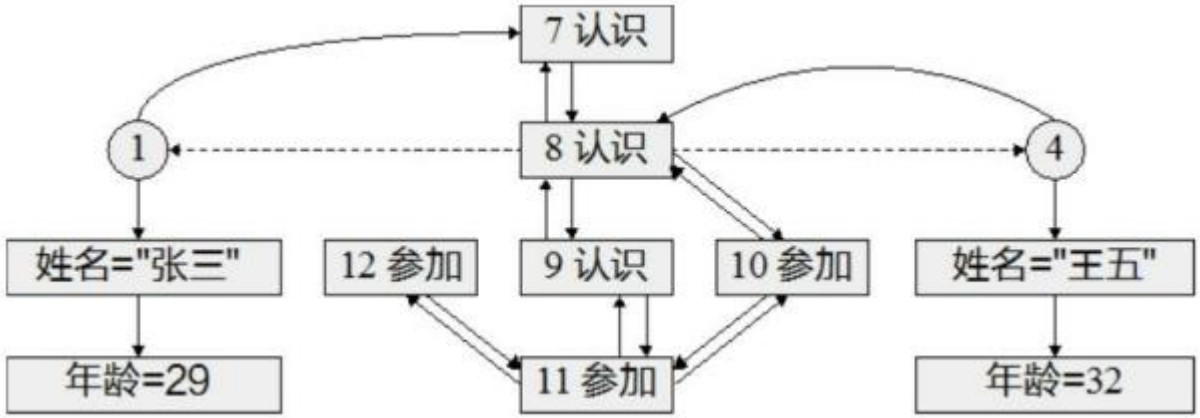
在 Neo4j数据库中，属性图的不同部分是被分开存储在不同文件中的。正是这种将图结构与图上属性分开存储的策略，使得 Neo4j 具有高效率的图遍历操作。

图数据库的文件存储结构

节点记录的第0字节inUse是记录使用标志字节的，告诉数据库该记录是否在使用中，还是已经删除并可回收用来装载新的记录；第1～4字节nextRelId是与节点相连的第1条边的id；第5～8字节nextPropId是节点的第1个属性的id。

第1～4字节 firstNode 和第5～8字节secondNode分别是该边的起始节点id和终止节点id；第9～12字节relType是指向该边的关系类型的指针；第13～16字节firstPrevRelId和第17～20字节firstNextRelId分别为指向起始节点上前一个和后一个边记录的指针；第21～24字节secPrevRelId 和第25～28字节secNextRelId 分别为指向终止节点上前一个和后一个边记录的指针；第29～32字节nextPropId是边上的第1个属性的id。

图遍历的查询的物理实现



由节点1导航到节点4的过程为：

（1）由节点1知道其第1条边为边7；

（2）在边文件中通过定长记录计算出边7的存储地址；

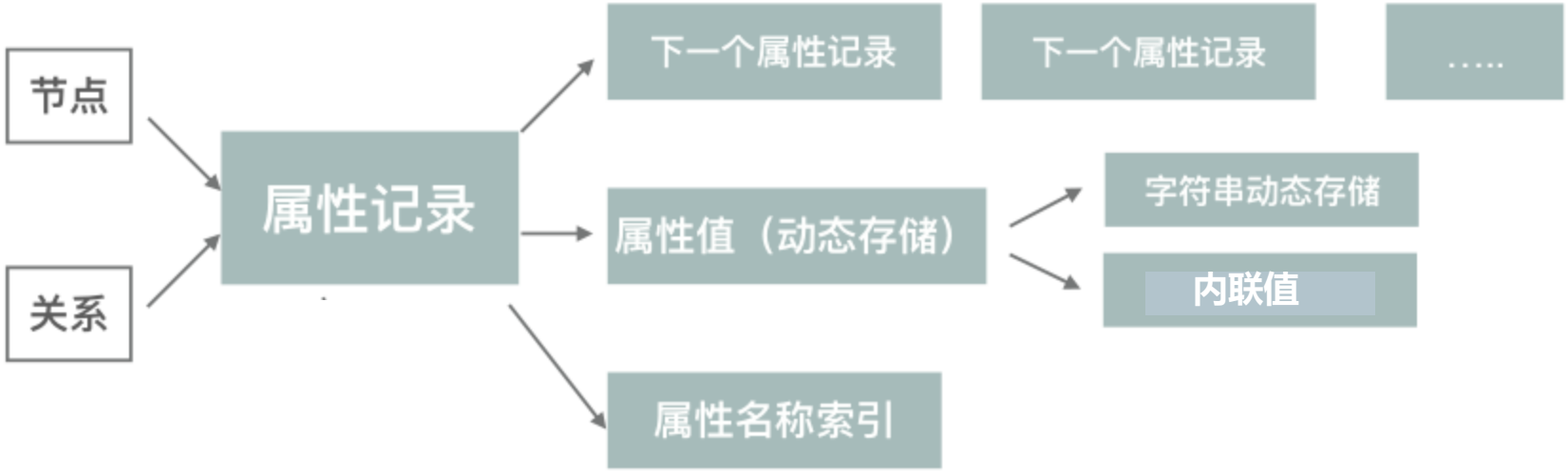
（3）由边7通过双向链表找到边8；

（4）由边8获得其中的终止节点id（secondNode），即节点4；

（5）在节点文件中通过定长记录计算出节点4的存储地址。

属性数据的存储处理： 内联与动态存储

- 图数据库中存在大量属性，这些属性的检索与图遍历的计算是分开的，这种设计是为了让节点之间的图遍历能不受大量属性数据的影响。
- 节点和关系的存储记录都包含指向它们的第一个属性ID的指针，与节点存储一样，属性记录也是固定大小，便于之间通过ID计算获得存储位置。
- 每个属性记录包含多个属性块，以及属性链中下一个属性的ID。
- 每个属性记录包含属性类型以及属性索引文件，属性索引文件存储属性名称。
- 对于每一个属性值，记录包含一个指向动态存储记录的指针（大属性值）或内联值（小属性值）。



知识图谱数据库的索引

- 一种是对节点或边上属性的索引，一种是对图结构的索引。
- 前者可以应用关系数据库中已有的B+树索引技术直接实现，后者仍是业界没有达成共识的开放问题。

1. 属性数据索引

- 以Neo4j为例，支持用户用Cypher语句对属性数据建立索引
- 例如，对程序员节点的姓名属性建立索引

```
CREATE INDEX ON : 程序员 (姓名)
```

```
MATCH (p:程序员{姓名：‘张三’} )
Return p
```

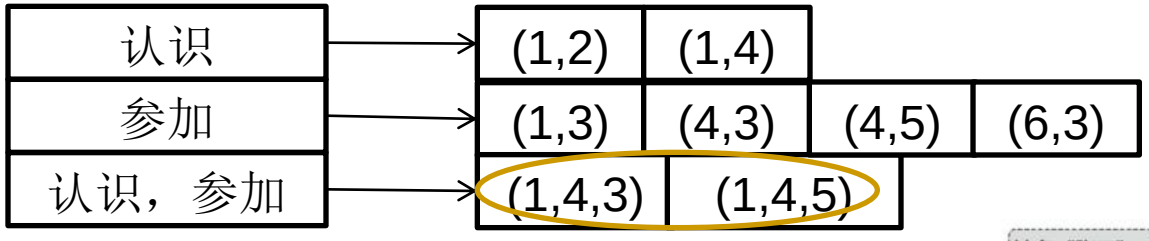
```
Drop Index on :程序员(姓名)
```

知识图谱数据库的索引

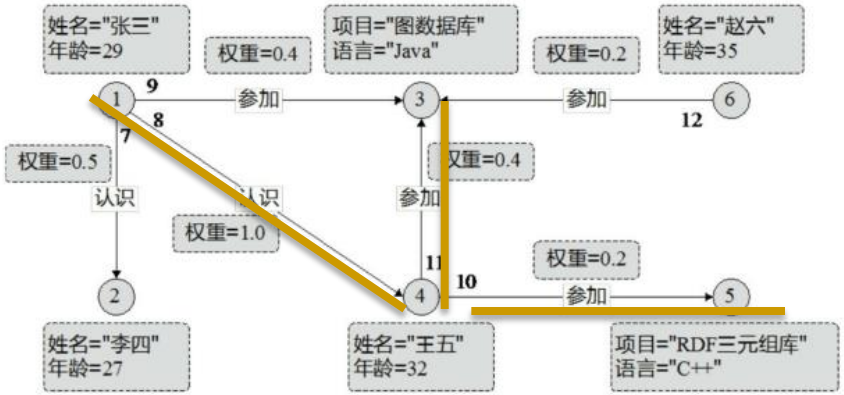
2. 图结构索引

- 为图数据中点边结构信息建立索引。利用图结构索引可以对图查询中的结构信息进行快速匹配，从而大幅消减搜索空间。
- 分为“基于路径”的和“基于子图的”方法

基于路径的图索引：GraphGrep方法，将图中长度小于或等于一个固定长度的全路径构建为索引结构。索引的关键字可以是组成路径的节点或边上属性值或标签的序列。



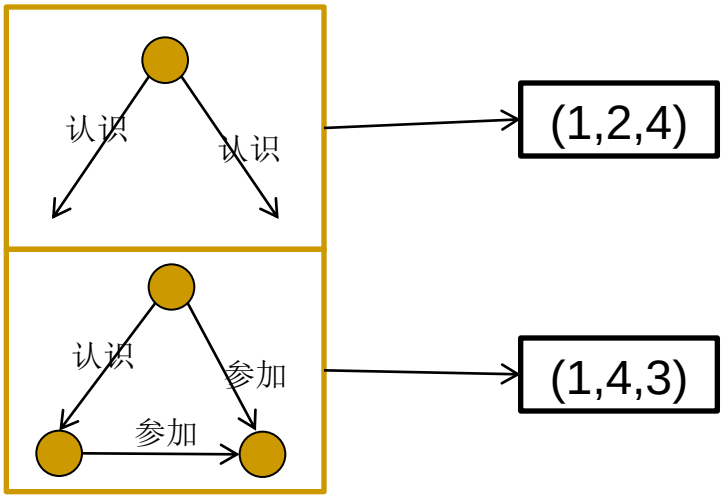
“查询年龄为29的参加了项目3的程序员参加的其他项目及其直接或间接认识的程序员参加的项目”

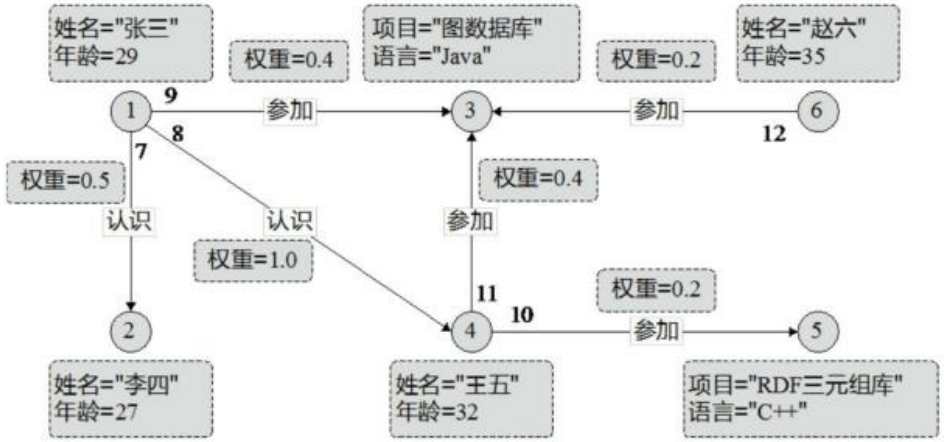


知识图谱数据库的索引

2. 图结构索引

基于子图的索引：将图数据中的某些子图结构信息作为关键字，将该子图的实例数据作为值构建索引结构。





如果查询中包含某些作为关键字的子图结构，则可以利用该子图索引，快速找到与这些子图匹配的节点序列。可大幅度检索查询操作的搜索空间。

总结

- 典型的知识存储引擎分为基于关系数据库的存储、面向RDF的三元组数据库和基于原生图的存储。
- 属性图是目前被图数据库业界采纳最广的一种图数据模型。
- RDF三元组数据库的查询语言SPARQL和属性图查询语言CYPHER或GREMLIN.
- 基于关系型数据库的知识图谱存储：三元组表、水平表、属性表、垂直划分、六重索引、DB2RDF.
- 图数据库存储原理：采用免索引邻接（Index-free adjacency）构建的图数据库引擎。
- 原生图数据库的物理存储实现。
- 原生图数据库的检索与索引。