第八讲 图存储与图计算 之

图数据的表示与存储

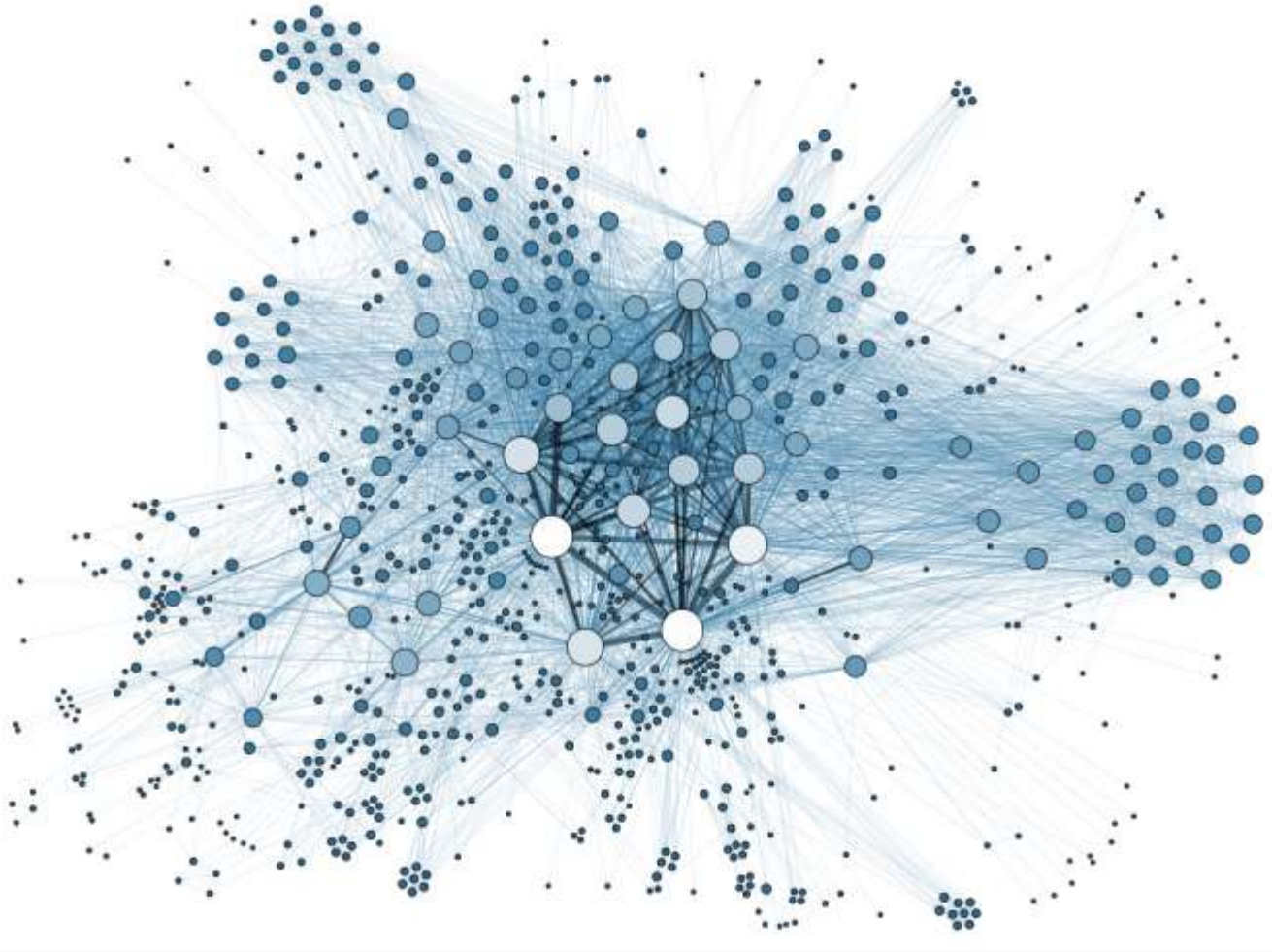
浙江大学计算机科学与技术学院 陈华钧 教授/博导

目录

第1节 图的基本知识

Network Science and Complex Network：

网络理论与复杂网络分析



社会学　　生物学

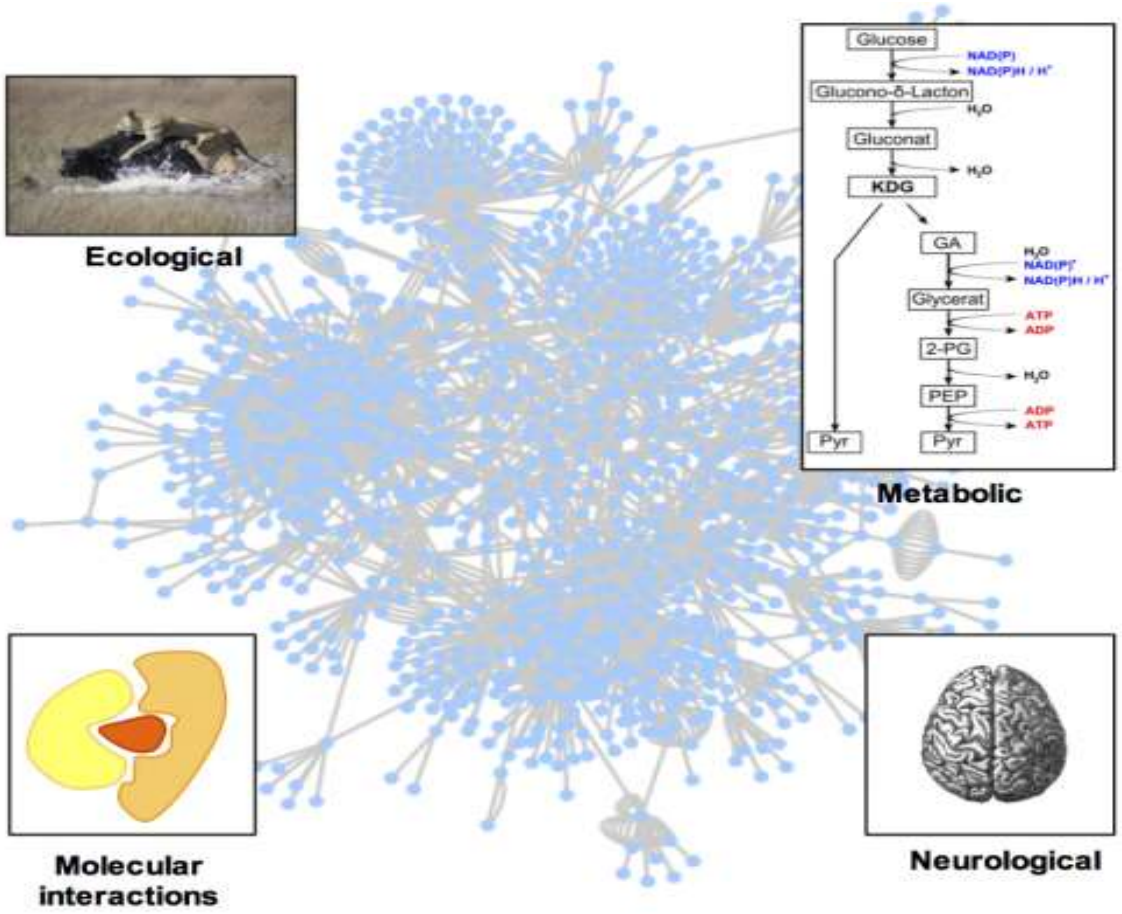
统计物理　　经济学

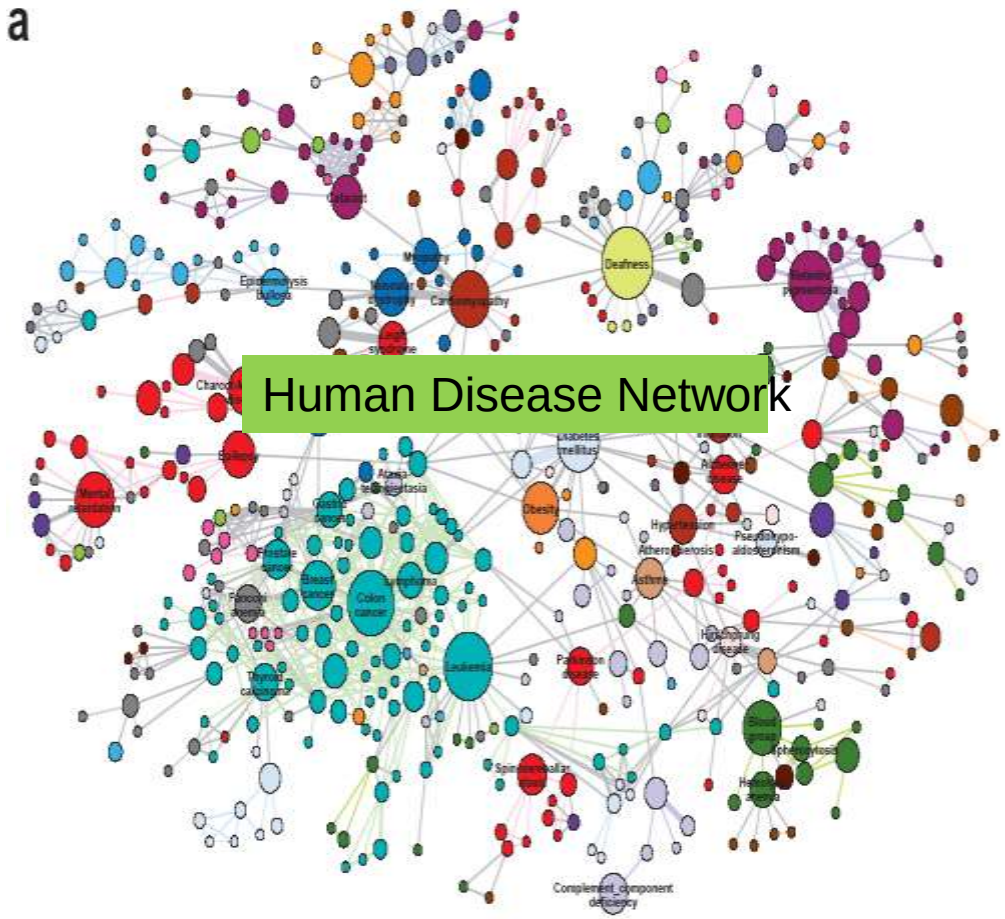
交通优化　　计算机

流行病学　　电气工程

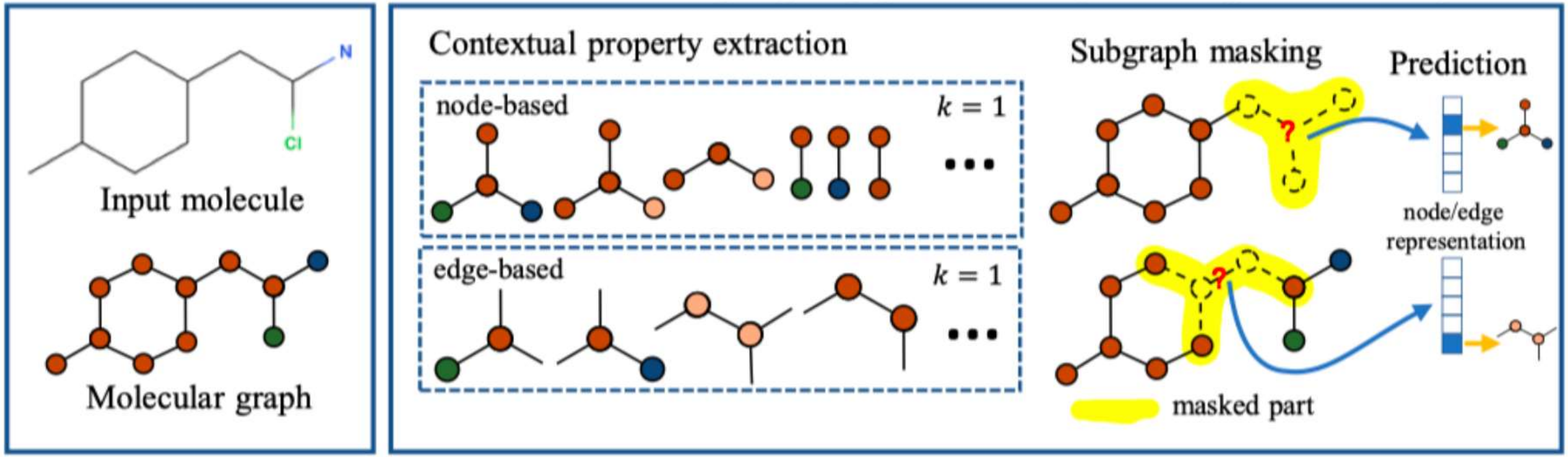
Network Science by Albert-László Barabási
http://networksciencebook.com/

图数据举例： Biological Network

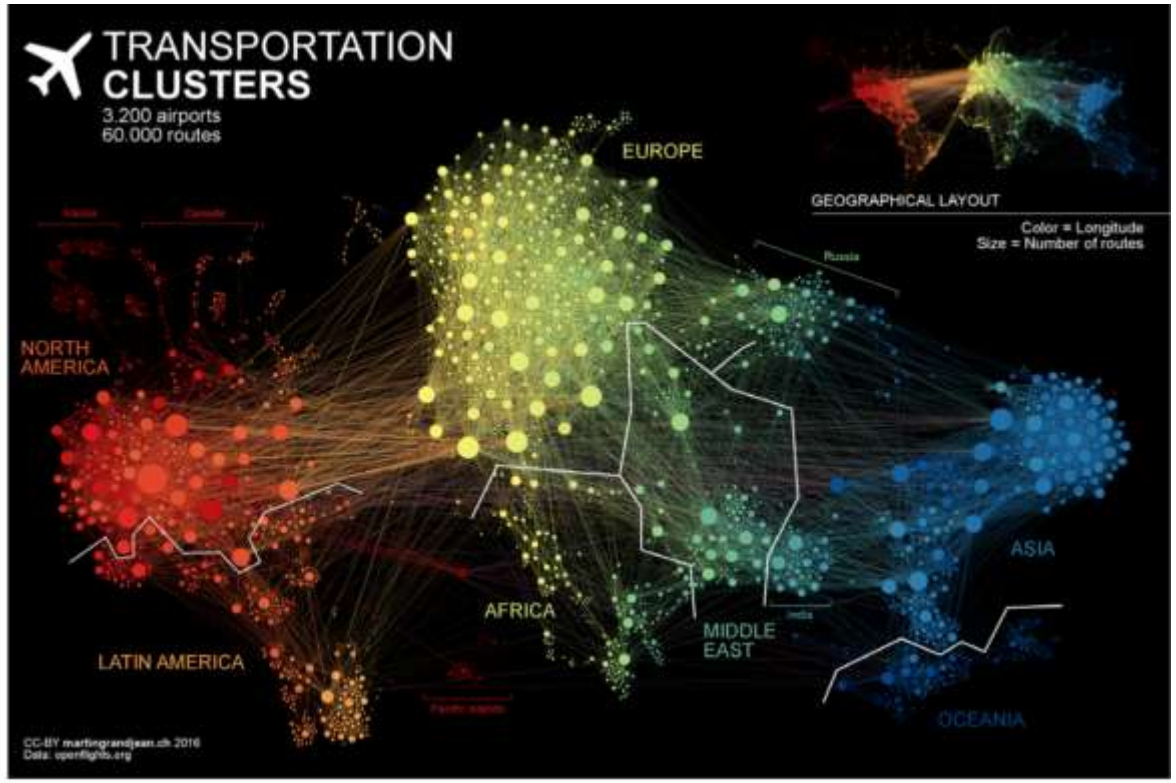




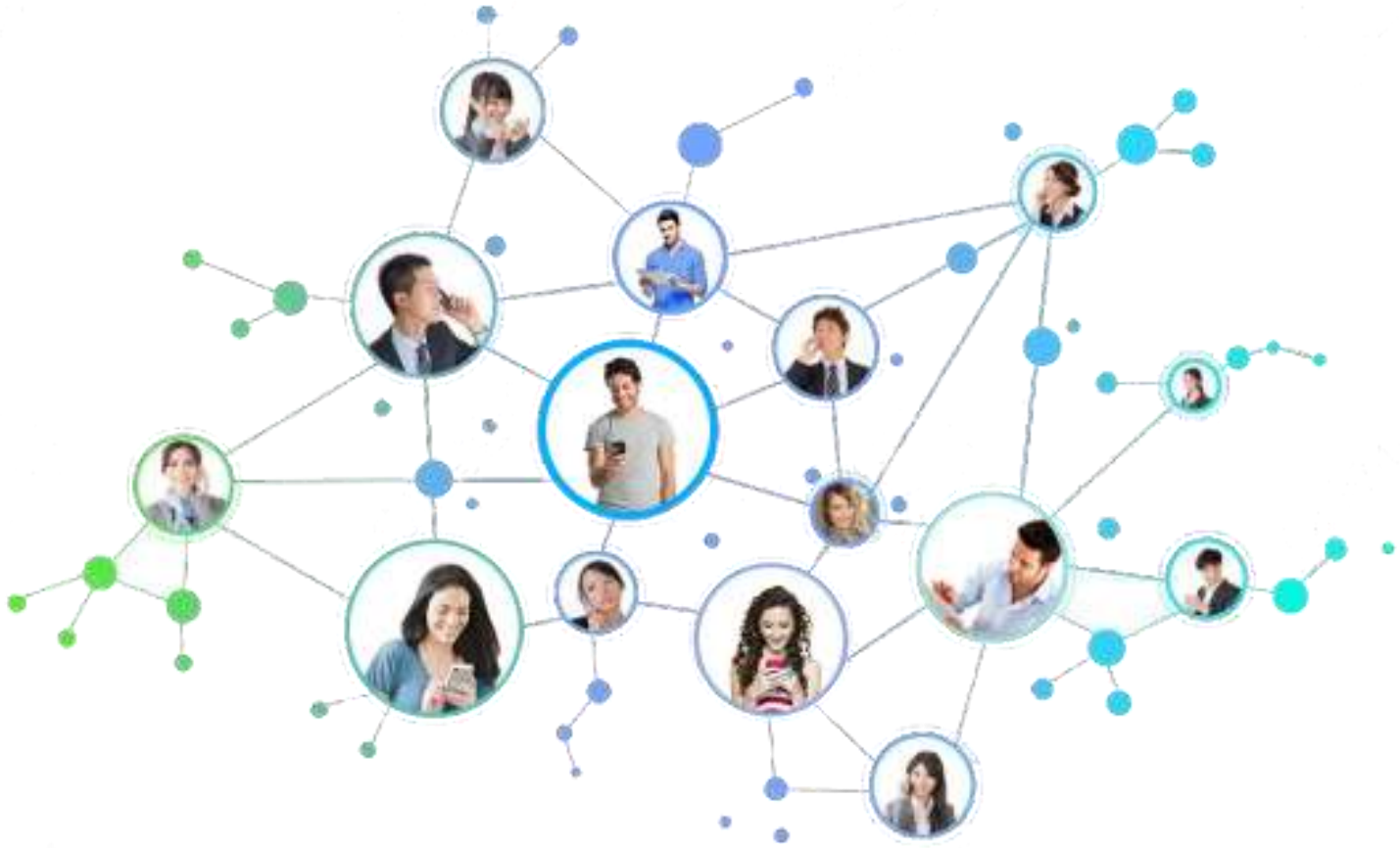
图数据举例： Molecular Graph



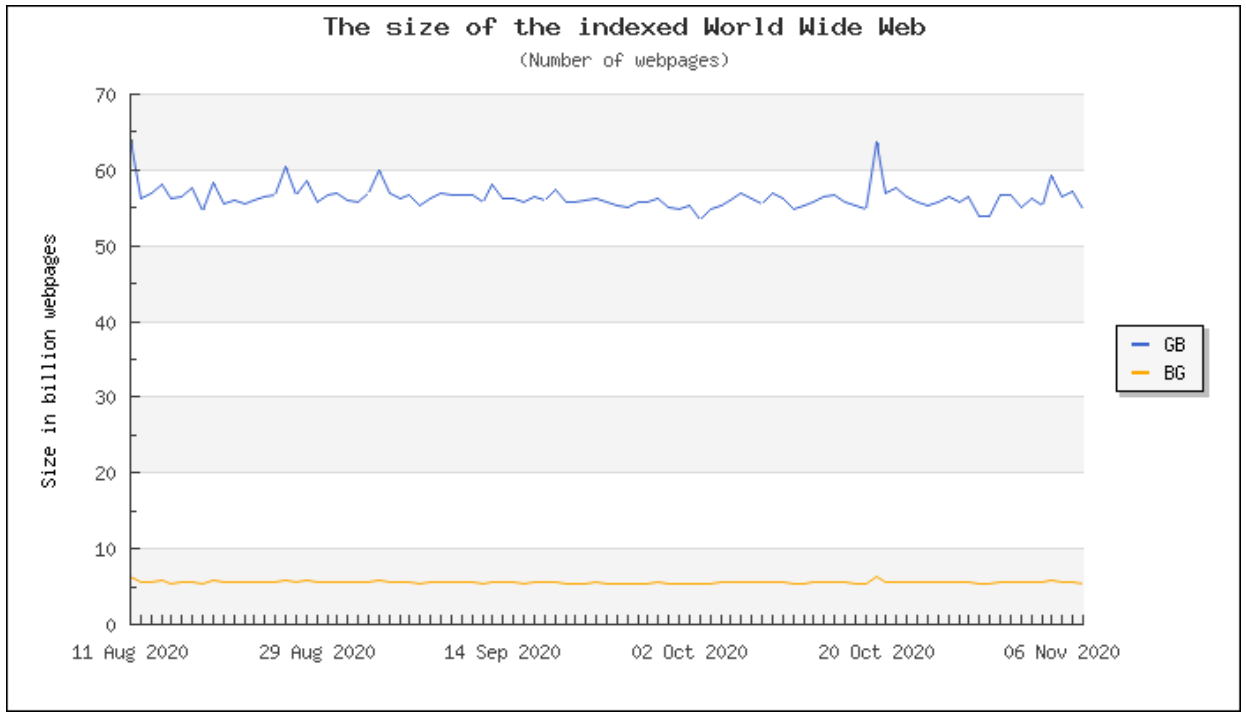
图数据举例： Transportation Network



图数据举例： Social Network



图数据举例： The Big Web



图数据举例：知识图谱

网页搜索

↓

语义搜索

WEB OF DOCS

↓

WEB OF DATA

手工众包 → 格式转化 元组抽取 实体融合 链接预测 推理补全 ← 语义嵌入

Freebase

社区协同构建

WIKIDATA

维基众包

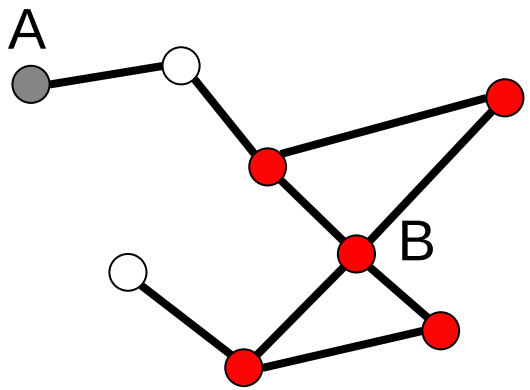
schema.org

....the new SEO?

网页嵌入语义数据

图的基本概念：Node Degree

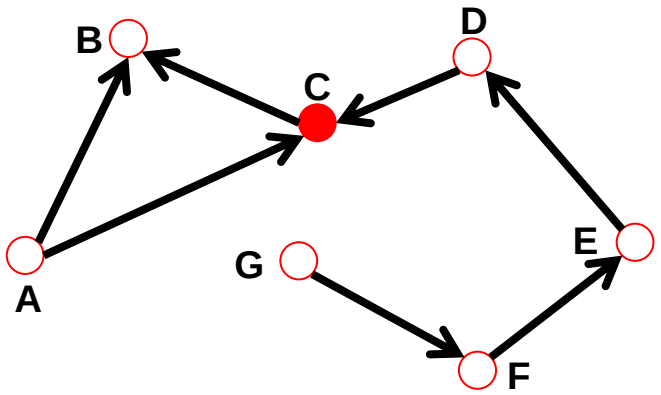
Undirected



Node degree: the number of links connected to the node.

$$k_A = 1 \qquad k_B = 4$$

Directed



In *directed networks* we can define an in-degree and out-degree. The (total) degree is the sum of in- and out-degree.

$$k_C^{in} = 2 \qquad k_C^{out} = 1 \qquad k_C = 3$$

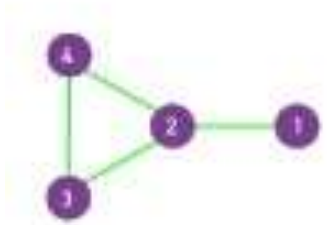
图的基本概念：Degree Dsitribution

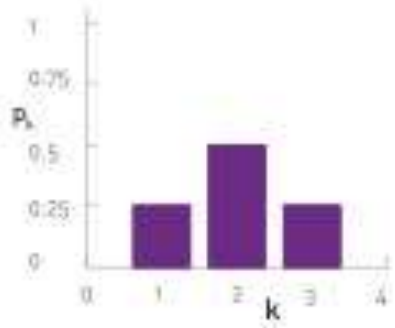
Degree distribution

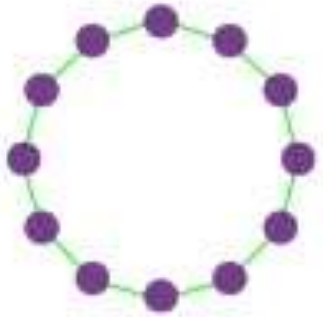
P(k): probability that a randomly chosen node has degree *k*

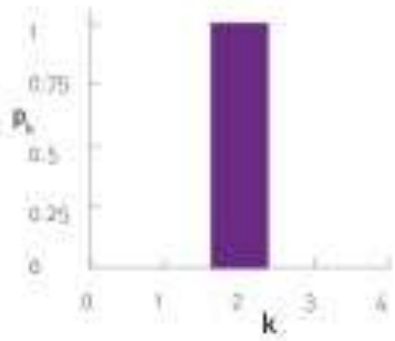
N_k = # nodes with degree k

$P(k) = N_k / N$ ❾ **plot**

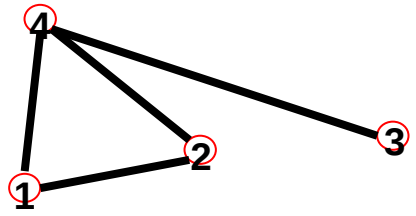








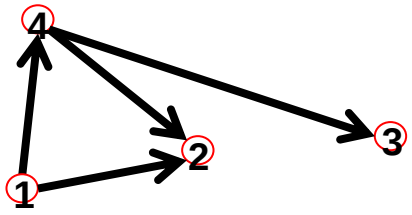
图的基本概念：邻接矩阵



$$A_{ij} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Aij=1 if there is a link between node i and j

Aij=0 if nodes i and j are not connected to each other.



$$A_{ij} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

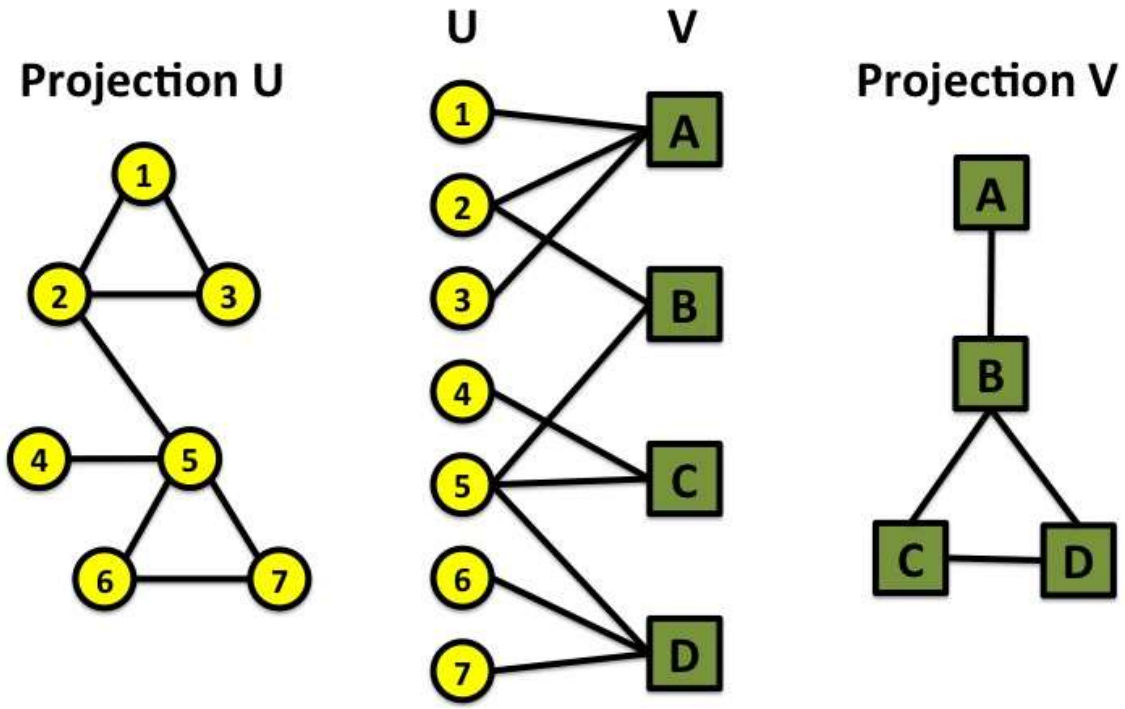
Note that for a directed graph (right) the matrix is not symmetric.

Aij=1 if there is a link pointing from node *j* and *i*

Aij=0 if there is no link pointing from *j* to *i*.

图的基本概念：bipartite graph

bipartite graph (or **bigraph**) is a graph whose nodes can be divided into two disjoint sets *U* and *V* such that every link connects a node in *U* to one in *V*; that is, *U* and *V* are independent sets.

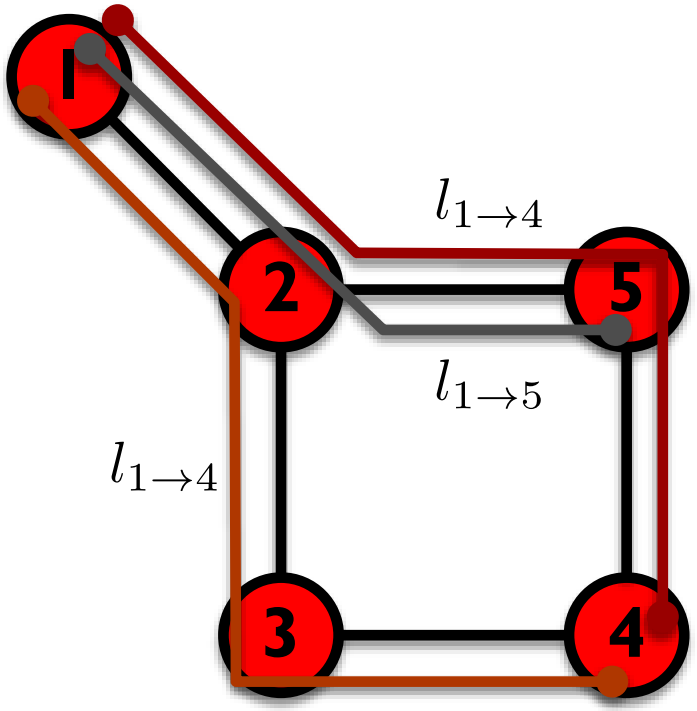


Examples:

Hollywood actor network
Collaboration networks
Disease network (diseasome)

图的基本概念：最短路径

Shortest Path



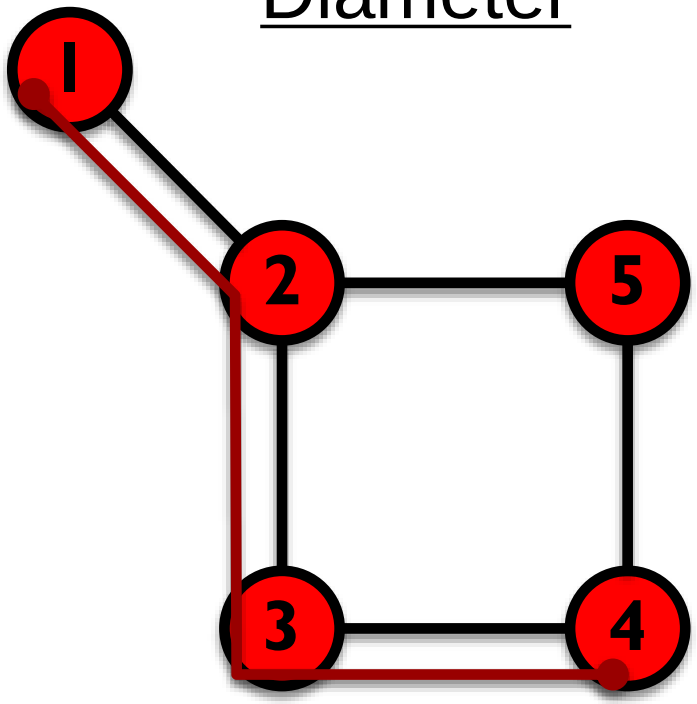
$$l_{1\to4} = 3$$

$$l_{1\to5} = 2$$

The path with the shortest length between two nodes (distance).

图的基本概念：图的直径与平均路径长度

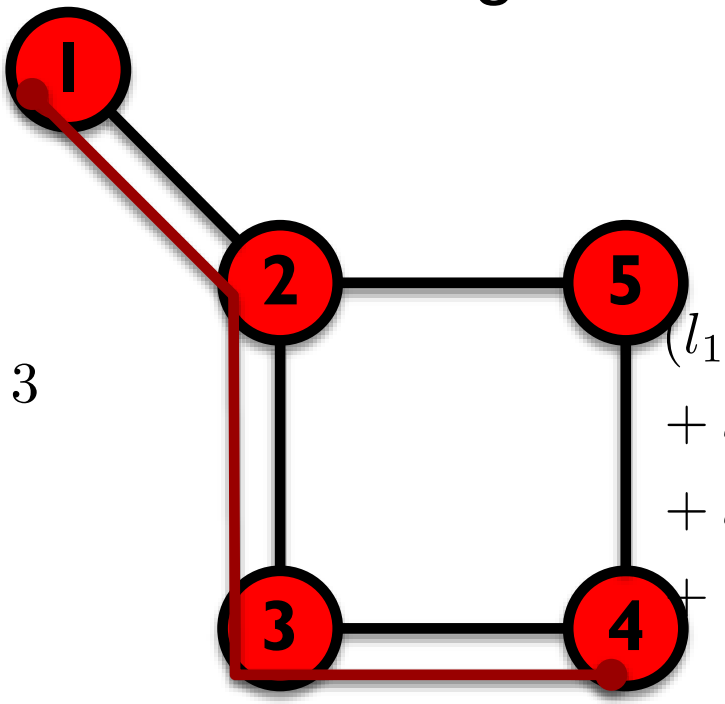
Diameter



$l_{1\to 4} = 3$

The longest shortest path in a graph

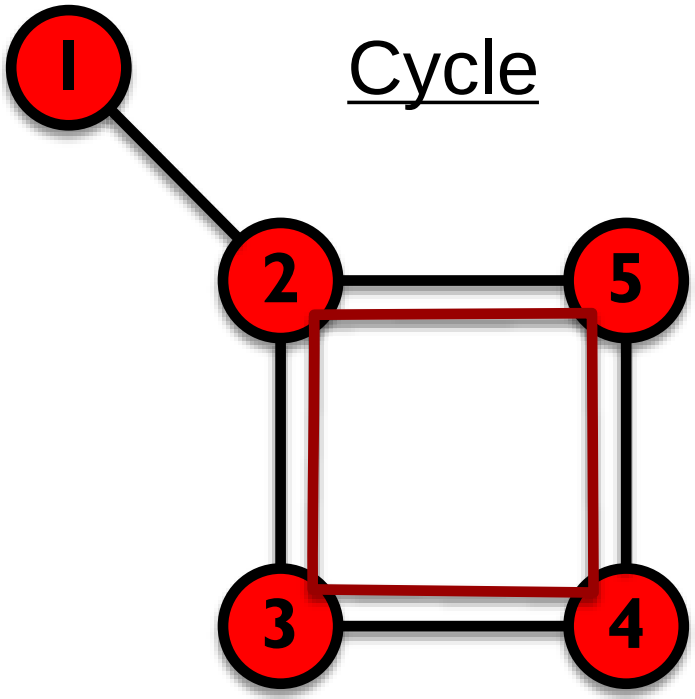
Average Path Length



$$(l_{1\to 2} + l_{1\to 3} + l_{1\to 4} + l_{1\to 5} + l_{2\to 3} + l_{2\to 4} + l_{2\to 5} + l_{3\to 4} + l_{3\to 5} + l_{4\to 5})/10 = 1.6$$

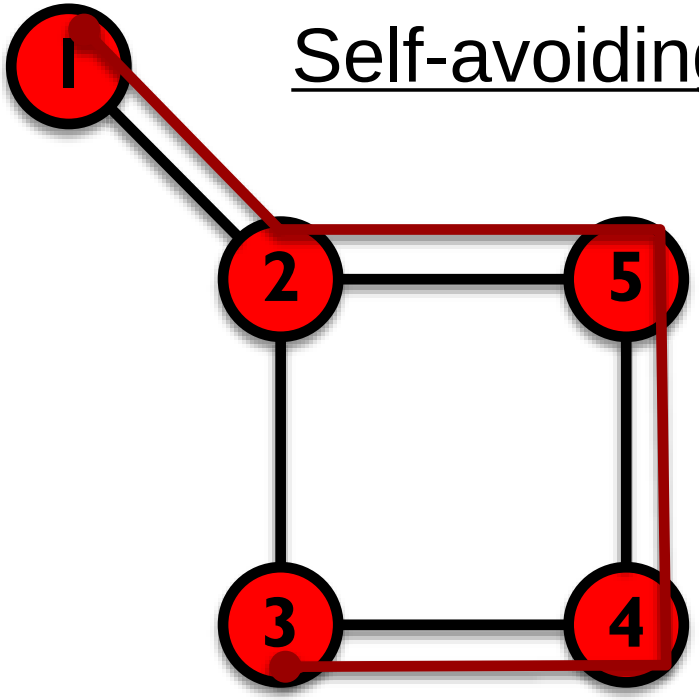
The average of the shortest paths for all pairs of nodes.

图的基本概念：Cycle和Self-avoiding Path



Cycle

A path with the same start and end node.

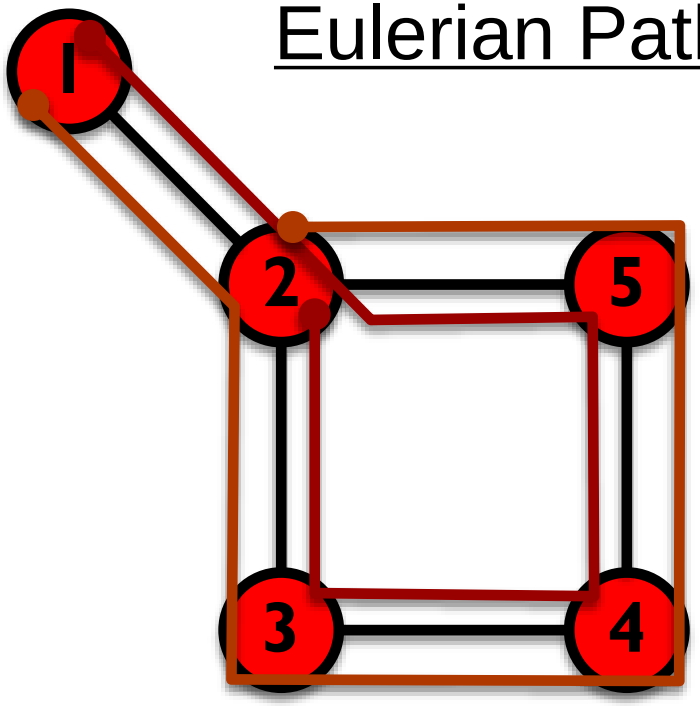


Self-avoiding Path

A path that does not intersect itself.

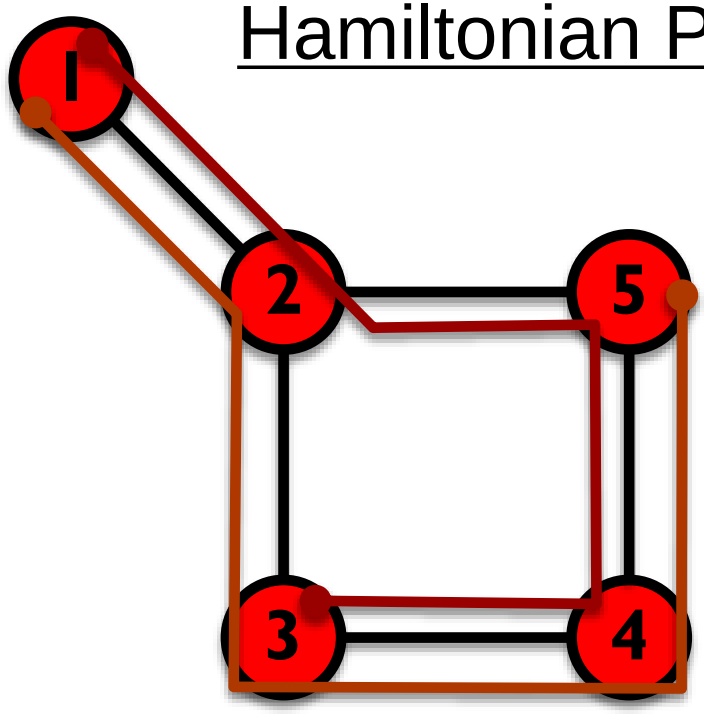
图的基本概念：欧拉路径和汉密尔顿路径

Eulerian Path



A path that traverses each link exactly once.

Hamiltonian Path



A path that visits each node exactly once.

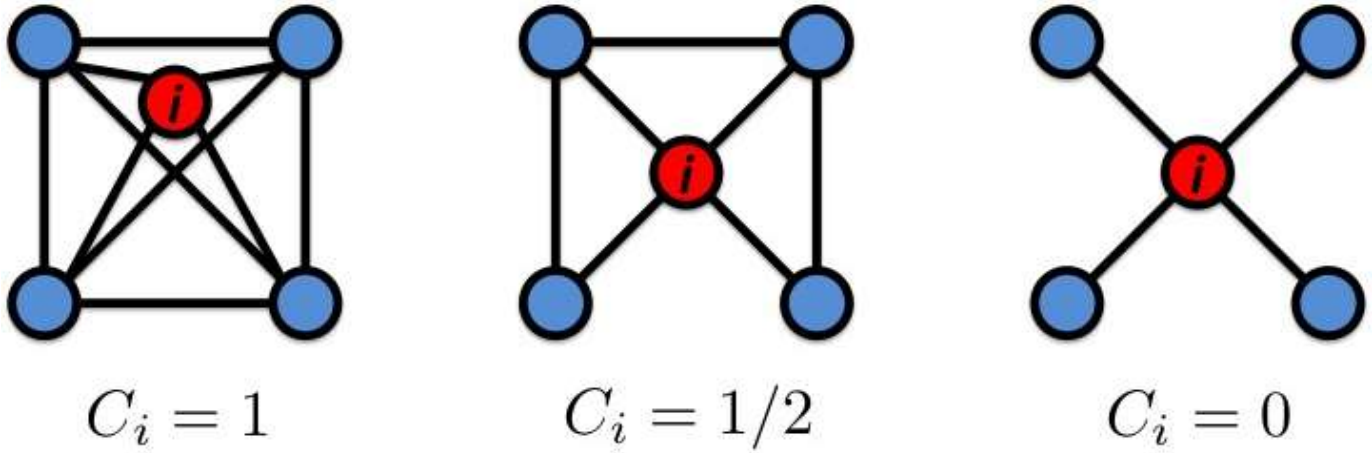
图的基本概念： Clustering coefficient

- **Clustering coefficient:**

  what fraction of your neighbors are connected?

  - Node i with degree k_i
  - C_i in [0,1]

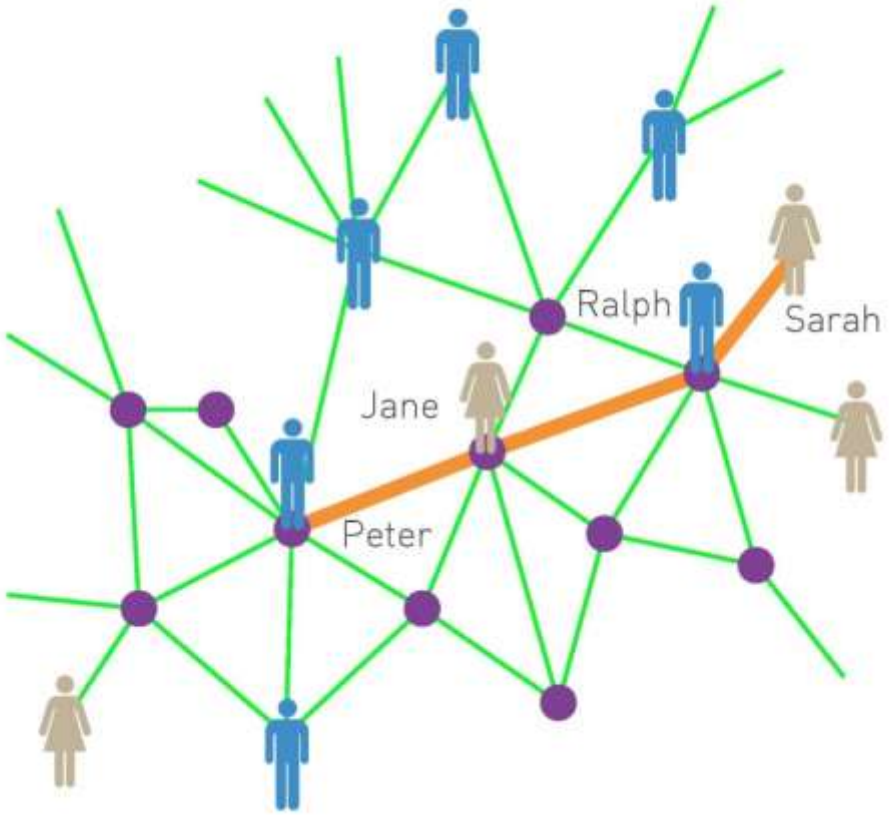
$$C_i = \frac{2e_i}{k_i(k_i - 1)}$$



$C_i = 1$ $C_i = 1/2$ $C_i = 0$

Watts & Strogatz, Nature 1998.

Graph Theory: Six Degree

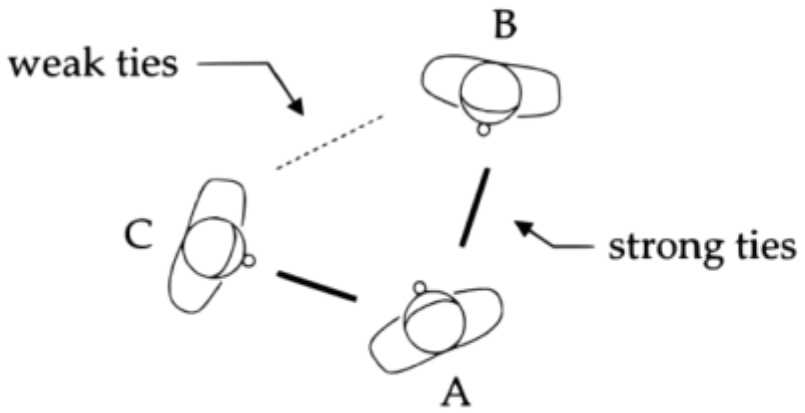


小世界模型

1. Stanley Milgram. The small world problem. Psychology Today, 2(1):60–67, 1967. Cited by 8500+ (as of Aug 2018)

Graph Theory: Weak Tie

- The **weak tie hypothesis** argues, if A is linked to both B and C, then there is a greater-than-chance probability that B and C are linked to each other.
- Essentially form "A friend's friend is also my friend"
- Another important hypothesis based on weak tie is that information diffusion through weak ties rather than srong ties.



Graph Theory: Scale-free Network Model

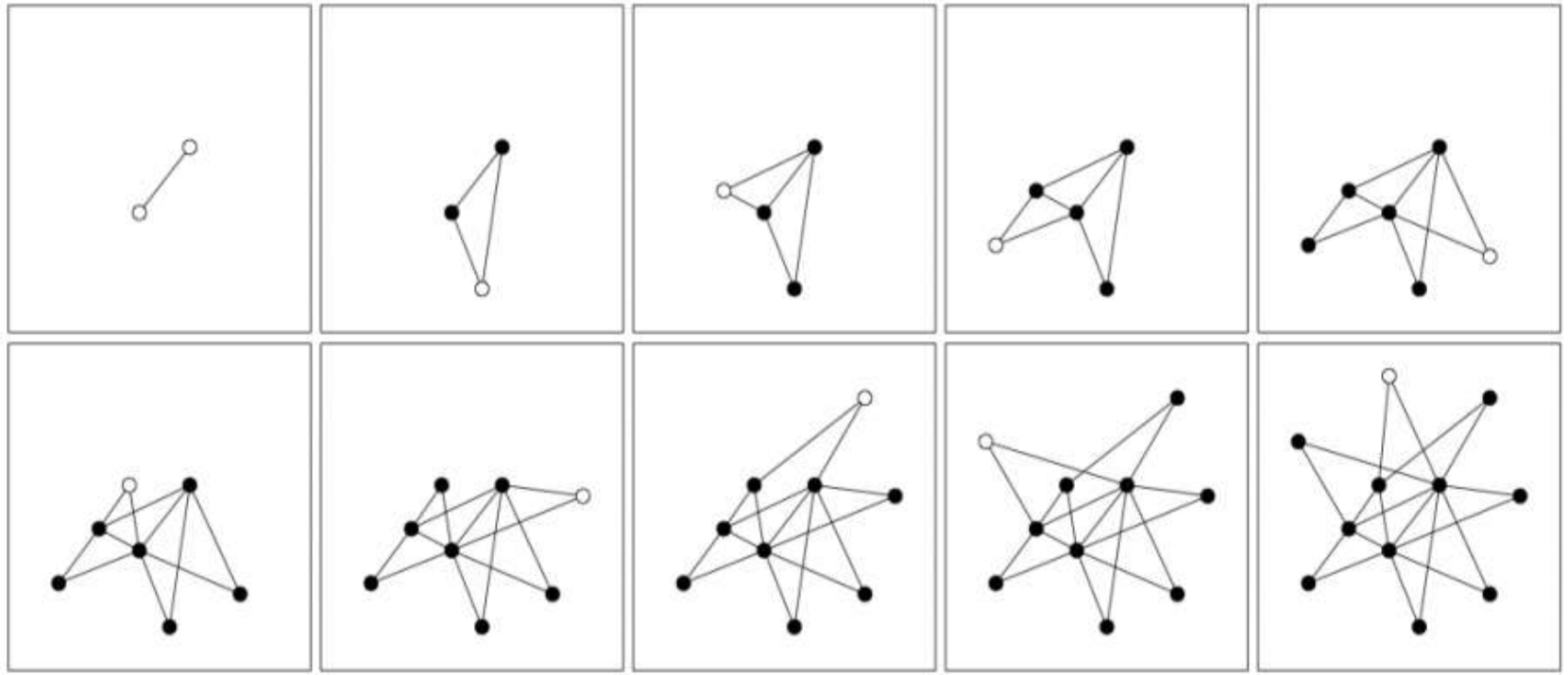
- 很多复杂系统拥有共同的重要特性:大部分节点只有少数几个连结，而某些节点却拥有与其他节点的大量连结。
- 这些具有大量连结的节点称为“集散节点”，所拥有的连结可能高达数百、数千甚至数百万。由此看来，这一特性似乎能说明网络是无尺度的。

网络	节点	连接
组织代谢	参与消化食物以释放能量的分子	参与相同的生化反应
好莱坞	演员	出演同一部电影
因特网	路由器	光纤及其它物理连接
蛋白质调控网络	协助调控细胞活动的蛋白质	蛋白质之间的相互作用
研究合作	科学家	合作撰写论文
性关系	人	性接触
万维网	网页	连接地址

1. Albert-László Barabási, Réka Albert. Emergence of scaling in random networks, Science, 286:509–512, 1999. Cited by 31000+ (as of Aug 2018)
2. Michalis Faloutsos, Petros Faloutsos, Christos Faloutsos. On power-law relationships of the internet topology. In ACM SIGCOMM 1999. Cited by 6400+ (as of Aug 2018)

Barabasi-Albert 模型：

优先连接-Preferential attachment



$$\Pi(k_i) = \frac{k_i}{\sum_j k_j}$$

Preferential attachment is a probabilistic mechanism: if a new node has a choice between a degree-two and a degree-four node, it is twice as likely that it connects to the degree-four node.

无尺度特性的产生主要来自于网络生成过程中的选择依附，新的节点会优先找已经有很多连接的节点进行连接，类似光环效应。

Barabasi-Albert 模型： 幂律分布

BA model is a random graph generation model using a preferential attachment mechanism: new nodes connect an existing node with the following probability:

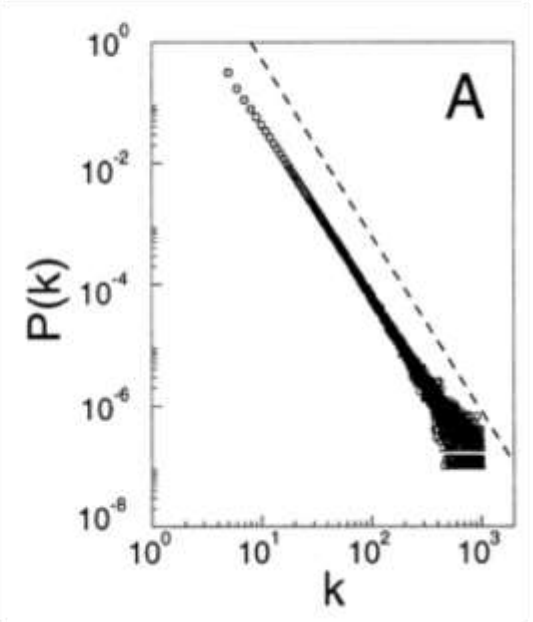
$$p_i = \frac{k_i}{\sum_j k_j}$$

- "A scale-free network is a network whose degree distribution follows a power law."

- The fraction *P(k)* of nodes having *k* connections to other nodes:

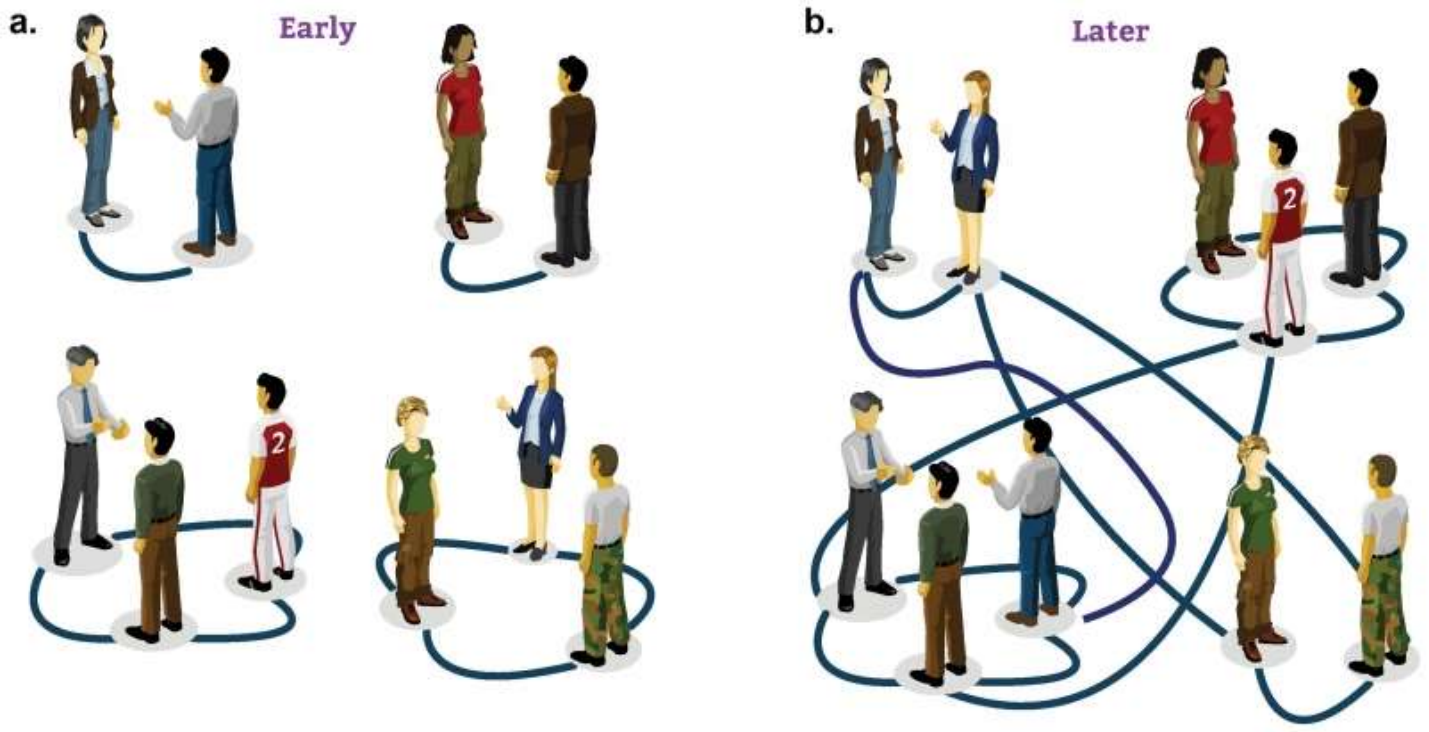
$$\boldsymbol{P(k) \sim k^{-\gamma}}$$

where γ is a parameter whose value is typically in the range $2 < \gamma < 3$

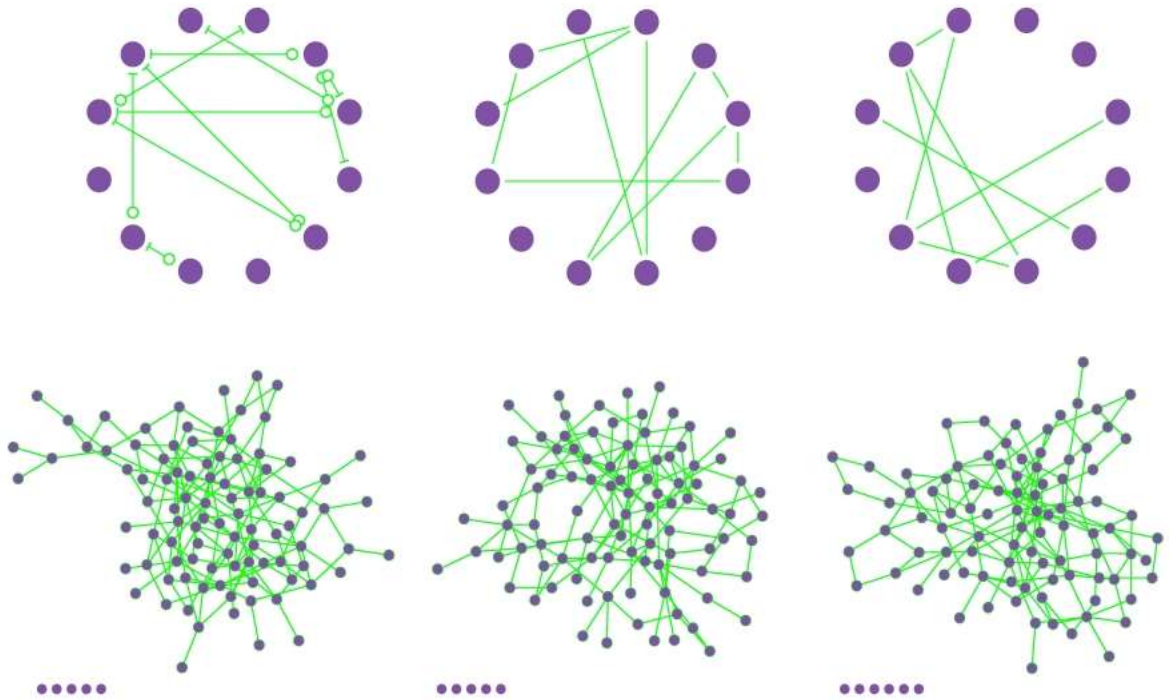


Graph Theory : Random Network Model

From a Cocktail Party to Random Networks

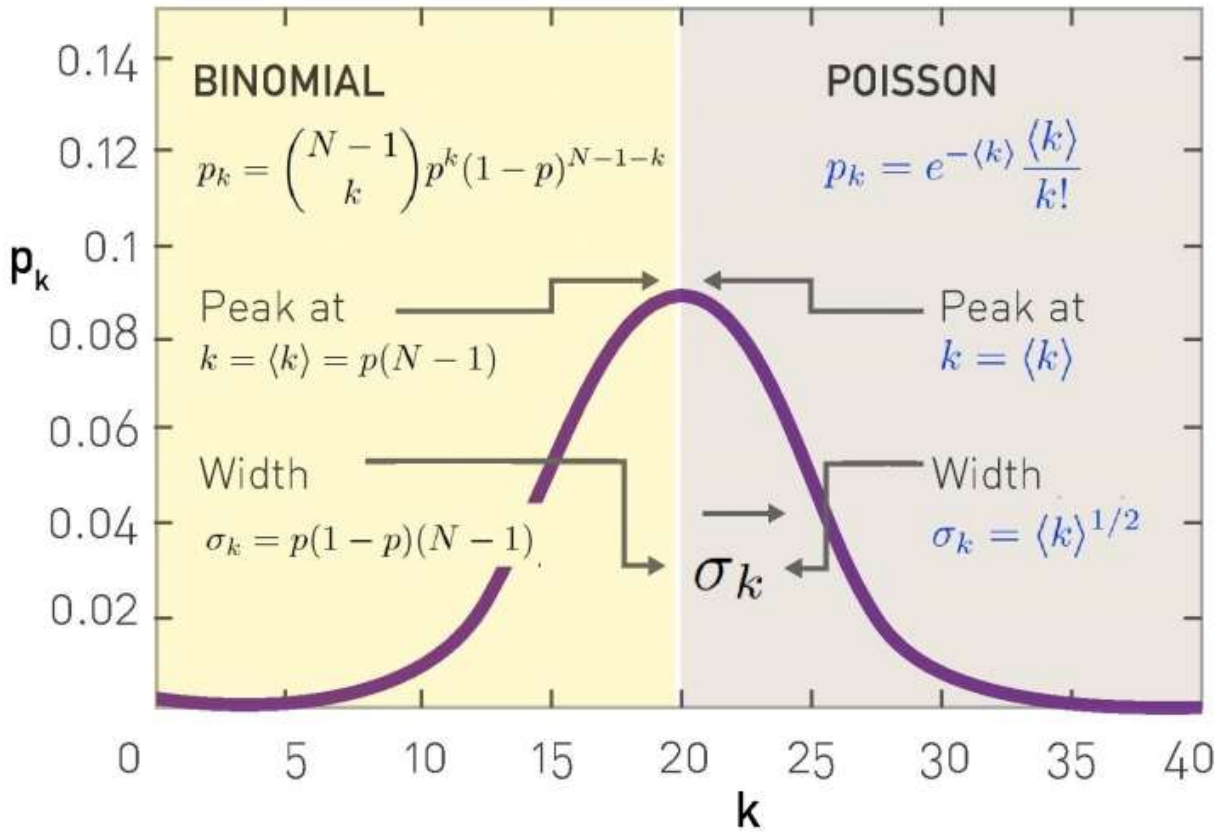


Erdos-Rényi模型： Random Network Model



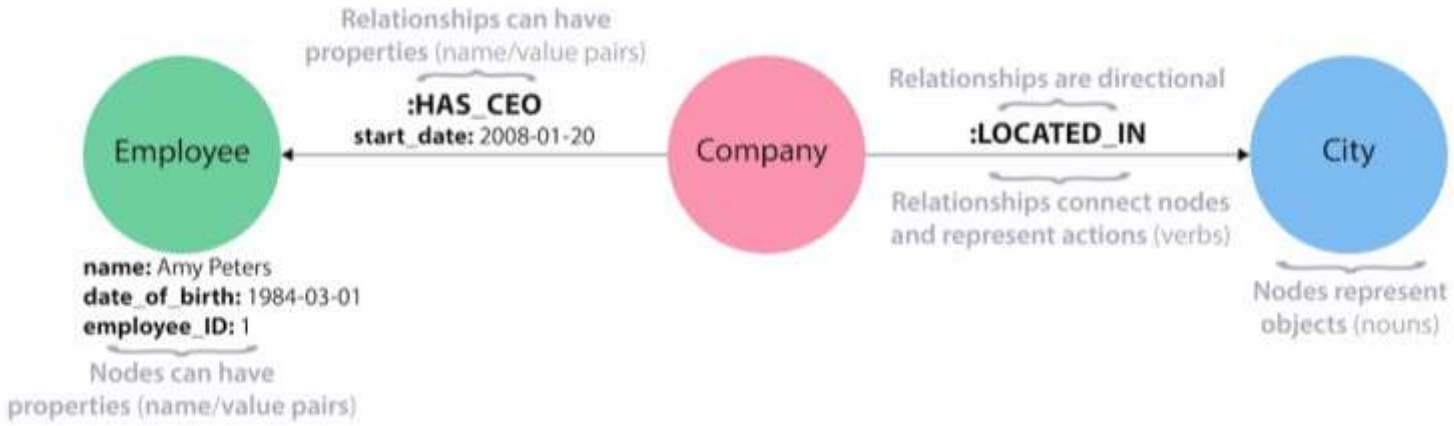
- 构建一个带有 n 个节点的随机图模型。这个图是通过以概率 p 独立地在节点 (i,j) 对之间画边来生成的。
- 因此，我们有两个参数：节点数量 n和概率 p。

Erdos-Rényi模型： Random Network Model



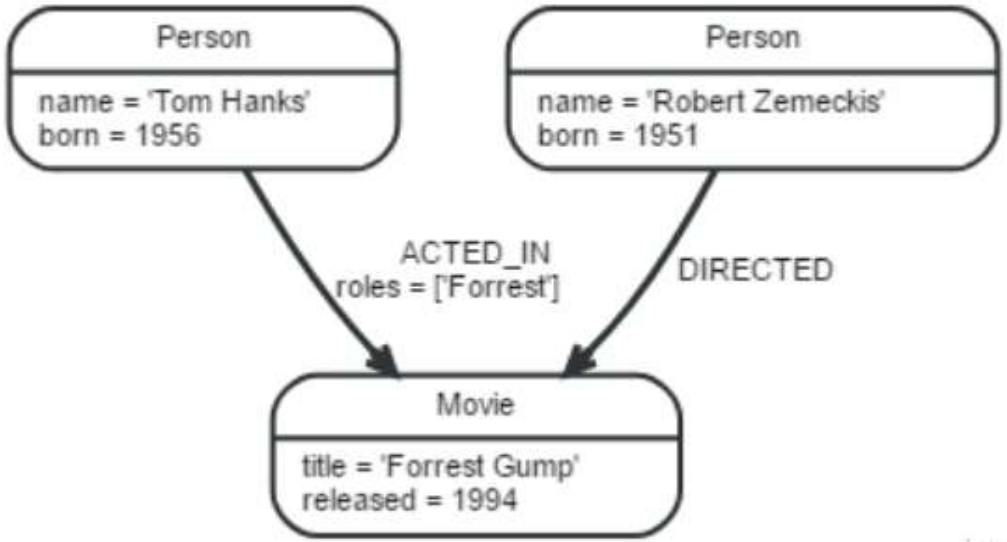
图数据的表示方法： Property Graph—属性图

- 属性图是图数据库Neo4J实现的图结构表示模型，在工业界有广泛应用。
- 属性图的优点是表达方式非常灵活，例如，它允许为边增加属性，非常便于表示多元关系。
- 属性图的存储充分利用图的结构进行优化，因而在查询计算方面具有较高优势。
- 属性图的缺点是缺乏工业标准规范的支持，由于不关注更深层的语义表达，也不支持符号逻辑推理。



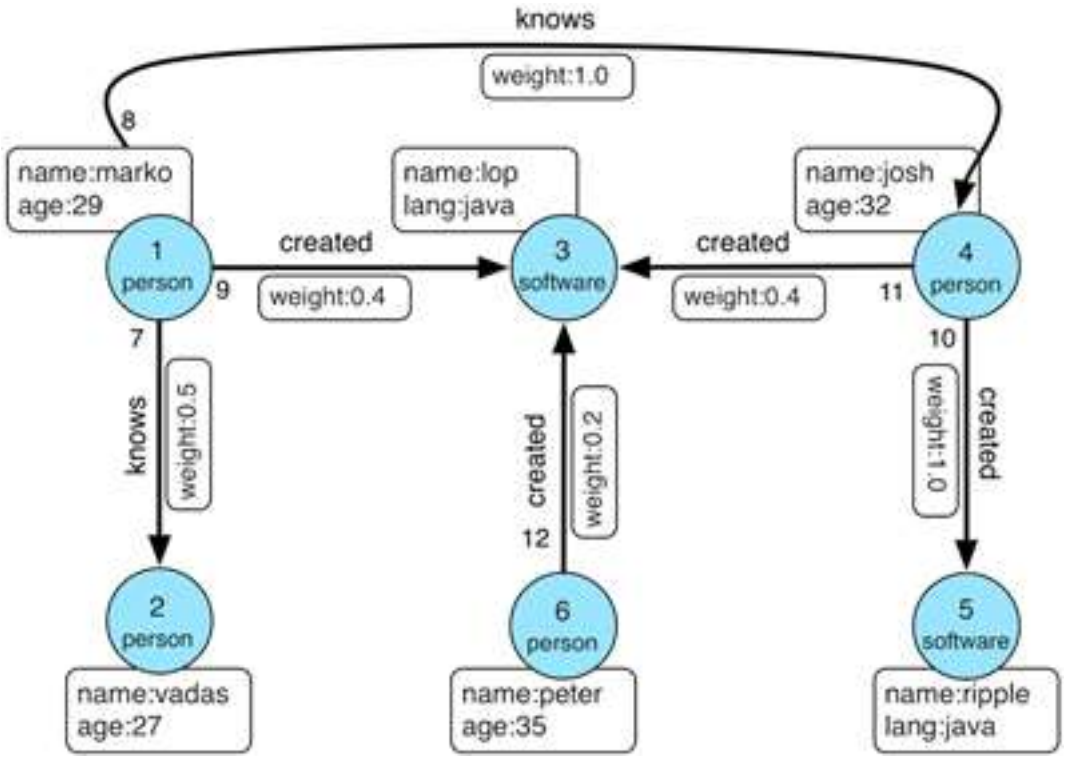
图数据的表示方法：Property Graph—属性图

- 在属性图的术语中，属性图是由 顶点（Vertex），边（Edge），标签（Label），关系类型还有属性（Property）组成的有向图。
- 顶点也称为 节点（Node），边也称为 关系（Relationship）。
- 在属性图中，节点和关系是最重要的实体。节点上包含属性，属性可以以任何键值形式存在。



图数据的表示方法：Property Graph—属性图

- 关系连接节点，每个关系都有拥有一个方向、一个标签、一个开始节点和结束节点。关系的方向的标签使得属性图具有语义化特征。
- 和节点一样，关系也可以有属性，即边属性，可以通过在关系上增加属性给图算法提供有关边的元信息，如创建时间等，此外还可以通过边属性为边增加权重和特性等其他额外语义。

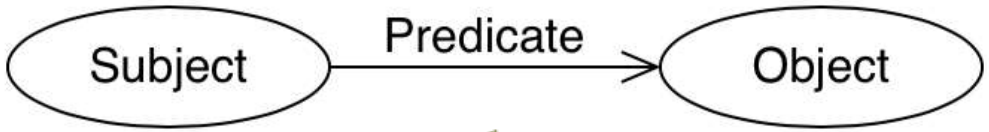


图数据的表示方法： RDF

RDF 代表 Resource Description Framework (资源描述框架)

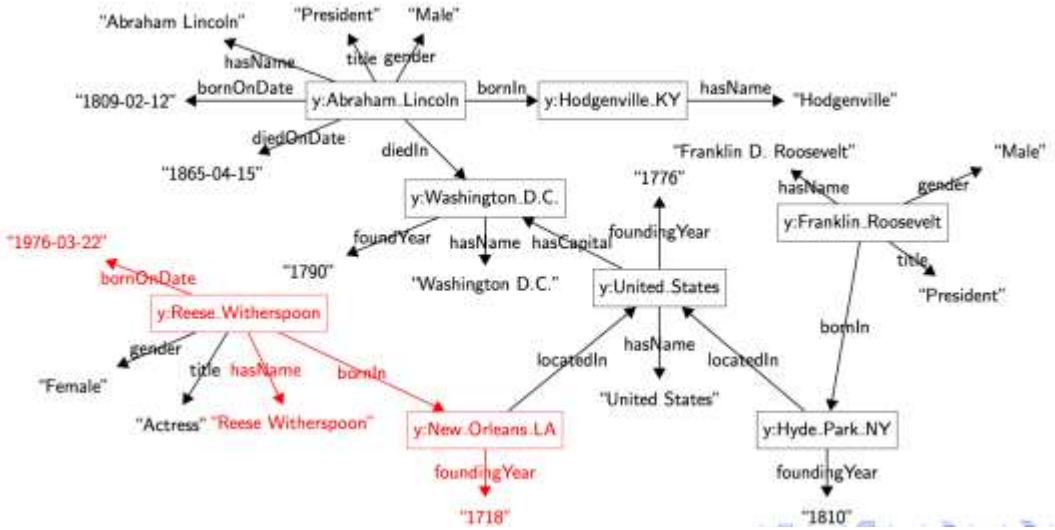
RDF是国际万维网联盟W3C推动的面向Web的语义数据标准

An RDF triple （S,P,O）encodes a statement—a simple ***logical expression***, or claim about the world.



(subject (主) , predicate (谓) , object (宾))

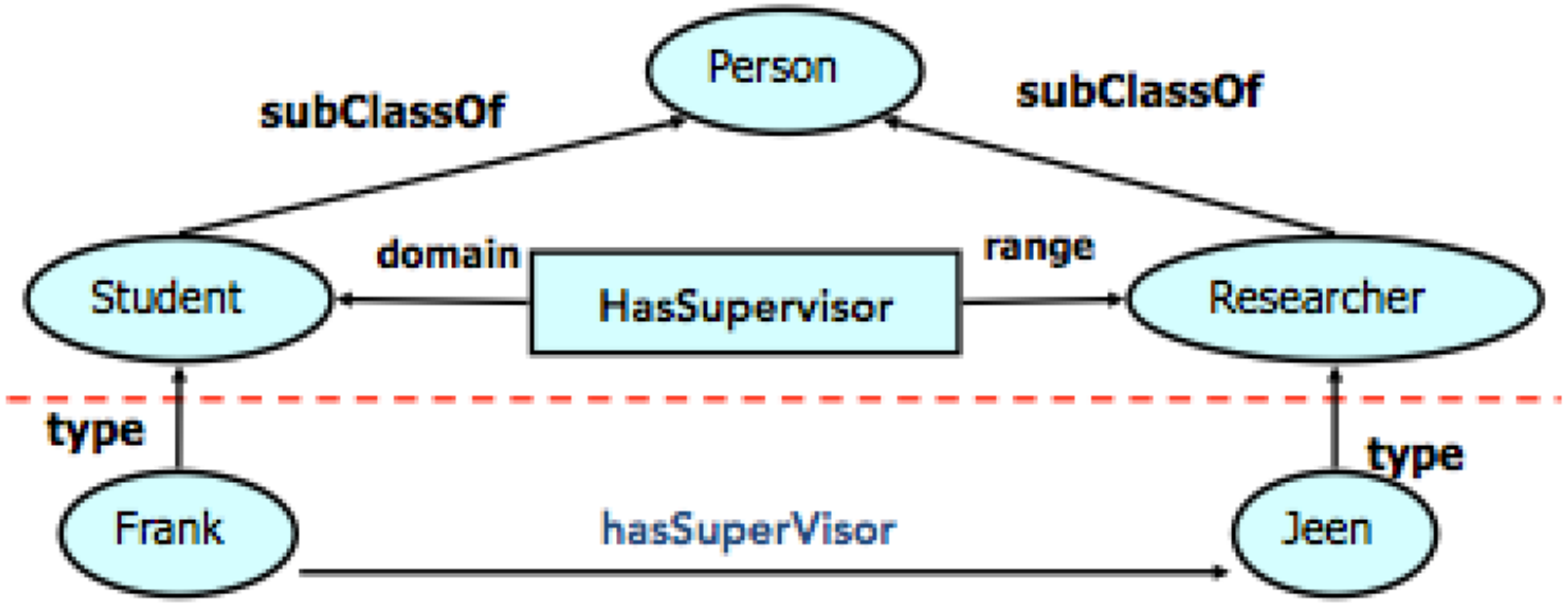
(subject (浙江大学) , predicate (位于) , object (杭州))



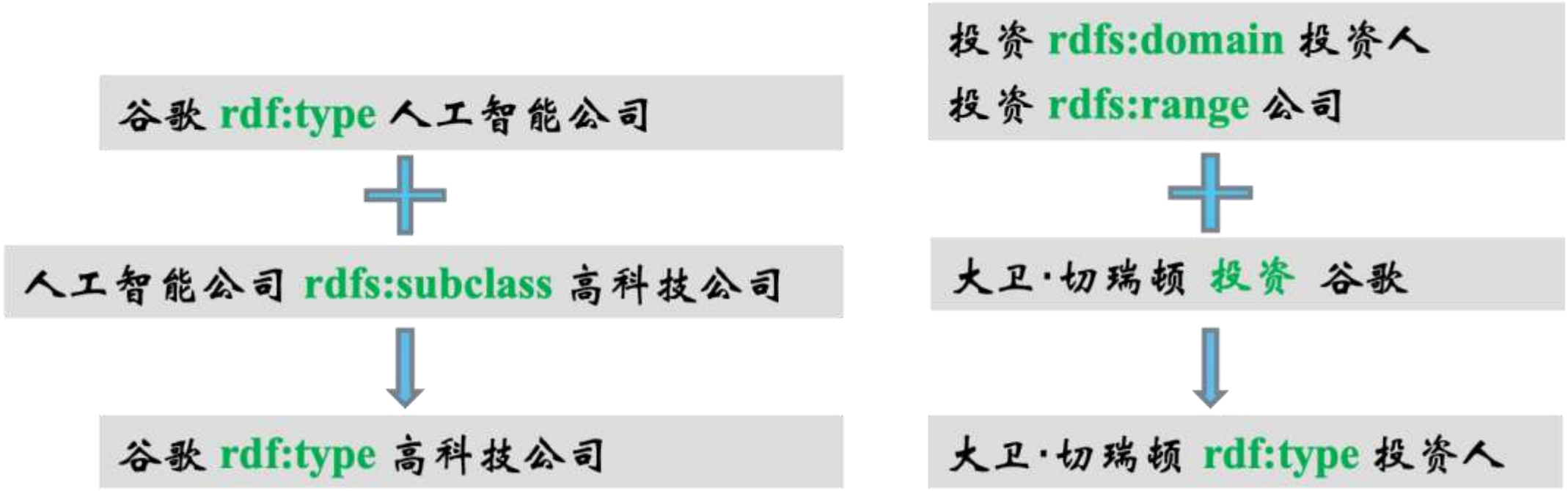
Triple-based Assertion model

RDFS: Simple Vocabulary and Schema

Defines small vocabulary for RDF:
Class, subClassOf, type, Property, subPropertyOf, Domain, Range



基于RDFS的简单推理



OWL extends RDF Schema

RDF-Schema： Class, subclass, Property, subProperty……

Complex Classes: intersection, union and complement

Property Restrictions: existential quantification, universal quantification, hasValue

Cardinality Restrictions: maxQualifiedCardinality, minQualifiedCardinality, qualifiedCardinality

Property Characteristics: inverseOf, SymmetricProperty, AsymmetricProperty, propertyDisjointWith, ReflexiveProperty, FunctionalProperty

Property Chains

OWL的表达构件

1. 等价性声明

exp:运动员 owl:equivalentClass exp:体育选手
exp:获得 owl:equivalentProperty exp:取得
exp:运动员A owl:sameIndividualAs exp:小明

2. 声明属性的传递性

exp:ancestor rdf:type owl:TransitiveProperty

exp:小明 exp:ancestor exp:小林;
exp:小林 exp:ancestor exp:小志

推理得出： exp:小明 exp:ancestor exp:小志 .

3. 声明两个属性互反

exp:ancestor owl:inverseOf exp:descendant
exp:小明 exp:ancestor exp:小林

推理得出：exp:小林 exp:descendant exp:小明

OWL的表达构件

4. 声明属性的函数性

exp:hasMother rdf:type owl:FunctionalProperty

exp:hasMother 是一个具有函数性的属性，因为每个人只有一个母亲，作为约束作用到知识库

5. 声明属性的对称性

exp:friend rdf:type owl:SymmetricProperty
exp:小明 exp:friend exp:小林

推理得出：exp:小林 exp:friend exp:小明

6. 声明属性的局部约束：全称限定

exp:Person owl:allValuesFrom exp:Women
exp:Person owl:onProperty exp:hasMother

exp:hasMother在主语属于exp:Person类的时候，宾语的取值只能来自exp:Women这个类。

OWL的表达构件

7. 声明属性的局部约束：存在限定

```
exp:SemanticWebPaper owl:someValuesFrom exp:AAAI
exp:SemanticWebPaper owl:onProperty exp:publishedIn

exp:publishedIn在主语属于exp:SemanticWebPaper类的时候，宾语的取值部分来自
exp:AAAI这个类。上面的三元组相当于：关于语义网的论文部分发表在AAAI上。
```

8. 声明属性的局部约束：基数限定

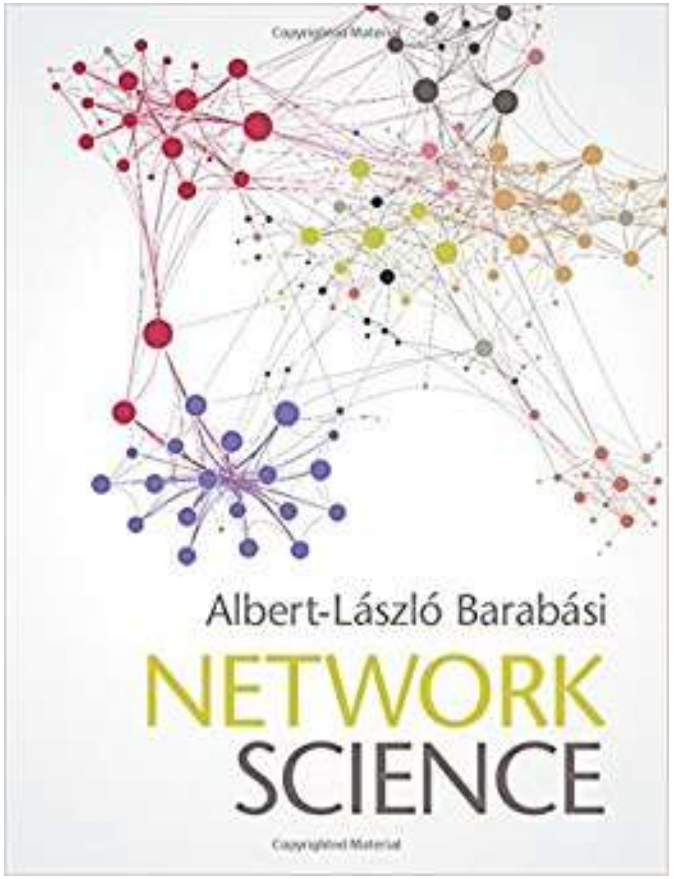
```
exp:Person owl:cardinality “1”^^xsd:integer
exp:Person owl:onProperty exp:hasMother

exp:hasMother在主语属于exp:Person类的时候，宾语的取值只能有一个；“1”的数据类型
被声明为xsd:integer；
这是基数约束，本质上属于属性的局部约束。
```

9. 声明相交的类

```
exp:Mother owl:intersectionOf _tmp
_tmp rdf:type rdfs:Collection
_tmp rdfs:member exp:Person
_tmp rdfs:member exp:HasChildren

_tmp是临时资源；它是rdfs:Collection类型，是一个容器；它的两个成员是
exp:Person，exp:HasChildren；上述三元组说明exp:Mother是exp:Person
exp:HasChildren这两个类的交集。
```

小结

- 图能够建模很多客观世界的复杂问题，图技术在研究疾病交互网络、交通网络、社交网络等很多领域有重要的应用。
- 常见的图和网络模型有无尺度网络和随机网络等，它们分别被用来建模不同类型的复杂问题。
- 知识图谱可以看作是图的一种应用，但与普通的图还是有非常大的区别。图的一些理论和算法多可以用来处理和分析知识图谱数据。
- 属性图是工业界最常见的图谱建模方法，属性图数据库充分利用图结构特点做了性能优化，实用度高，但不支持符号推理。
- RDF是W3C推动的语义数据交换标准与规范，有更严格的语义逻辑基础，支持推理，并兼容更复杂的本体表示语言OWL。

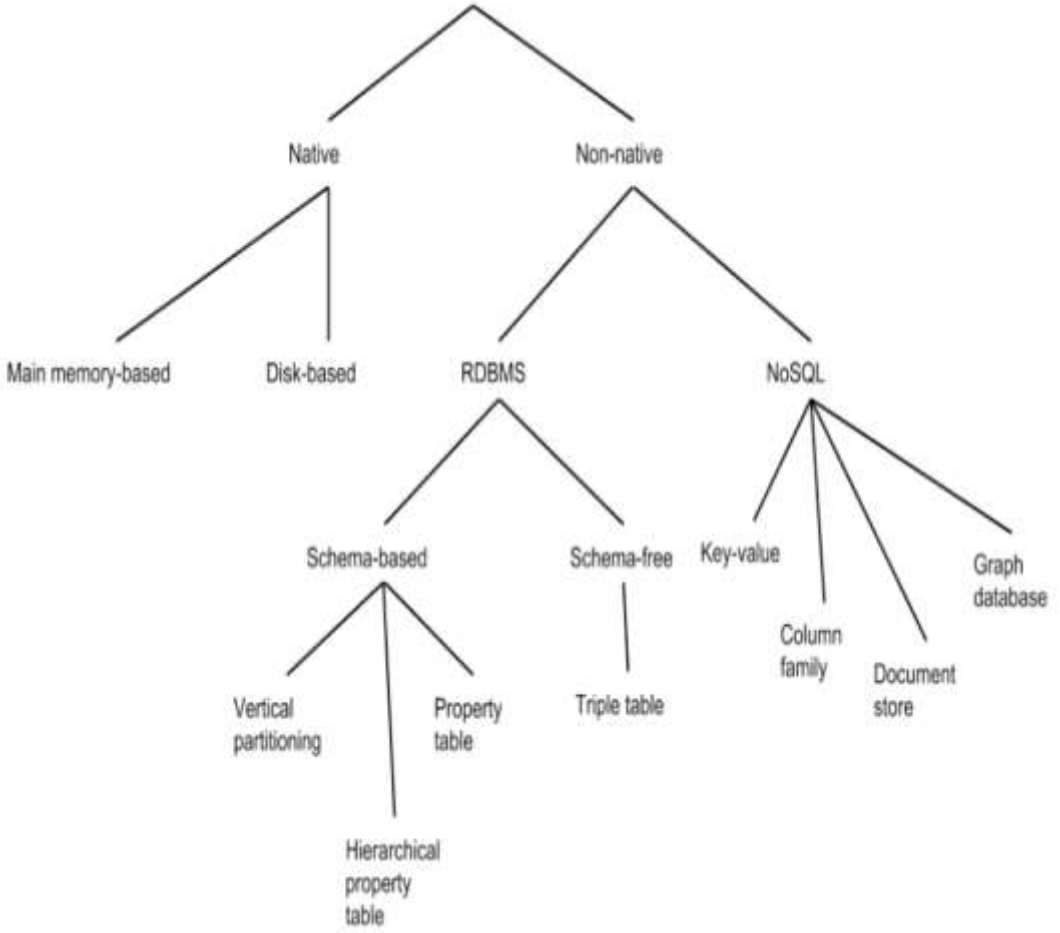


http://networksciencebook.com/

第2节 基于关系数据库的图数据存储

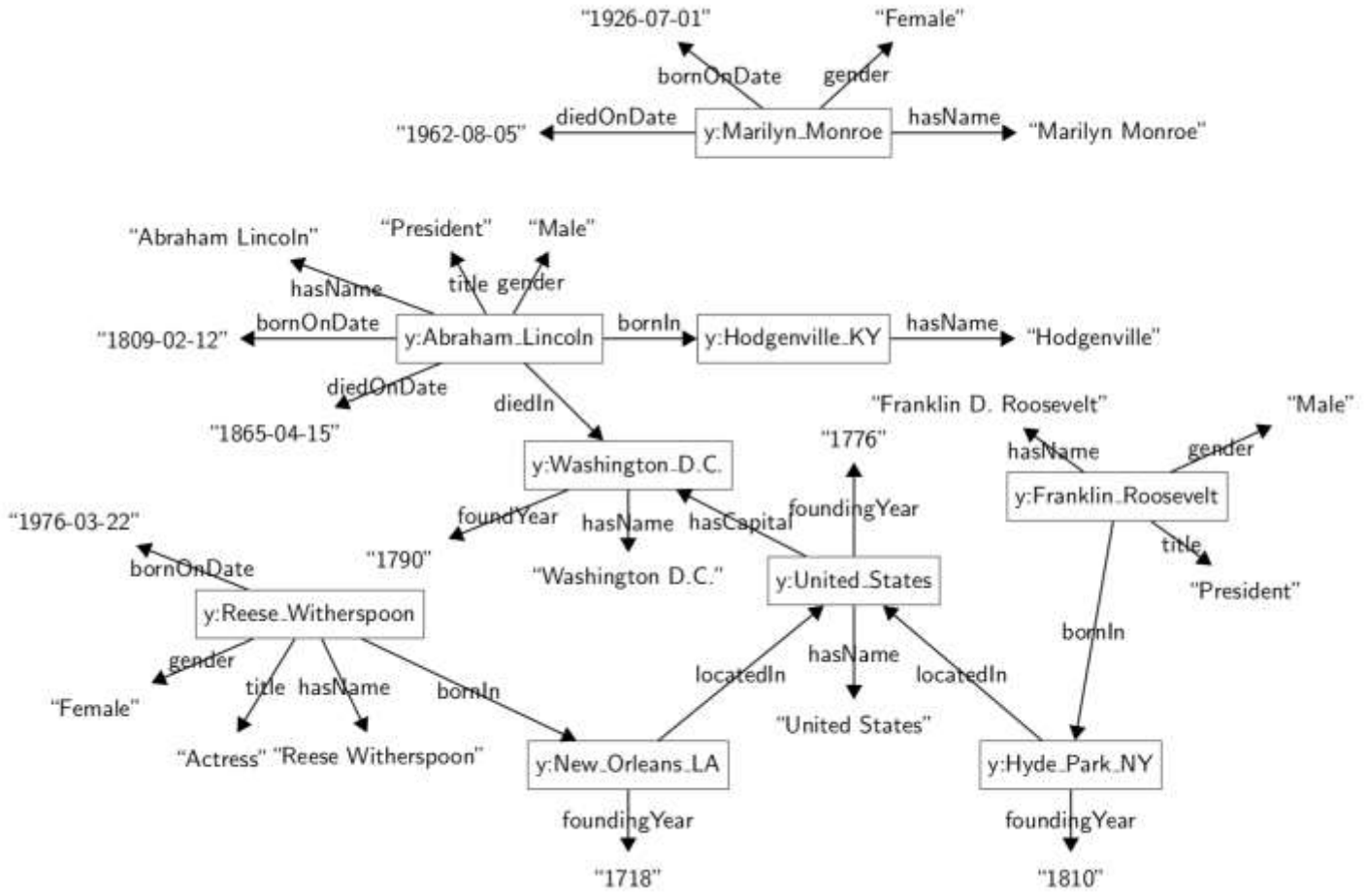
图数据的各种存储方式

- 图数据的存储需要综合考虑图数据结构、图的特点、索引和查询优化等问题。
- 典型的图数据存储引擎分为基于关系数据库的存储和基于原生图的存储。
- 图数据库存储并非必须，例如Wikidata项目后端是MySQL实现的。



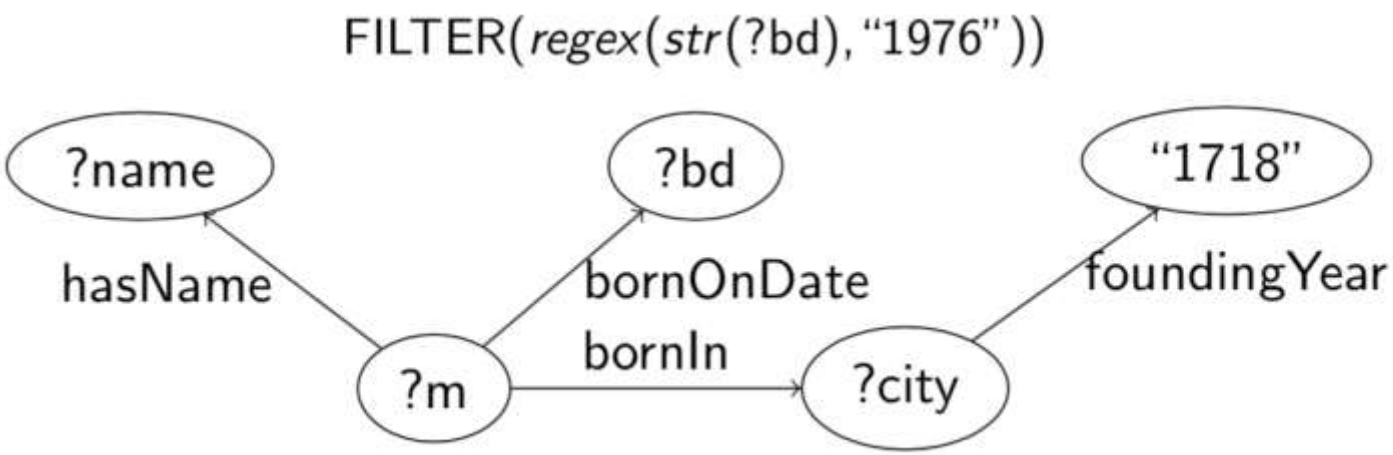
从图结构模型说起

属性图和RDF图模型都是有向标记图: Directed Labeled Graph



图上的查询语言：SPARQL

```
SELECT ?name
WHERE {
   ?m <bornIn> ?city. ?m <hasName> ?name.
   ?m<bornOnDate> ?bd. ?city <foundingYear> ''1718''.
   FILTER(regex(str(?bd),''1976''))
}
```



详细语法细节参考：https://www.w3.org/TR/sparql11-query/

最简单的存储： Triple Store

```
SELECT ?name
WHERE {
   ?m <bornIn> ?city. ?m <hasName> ?name.
   ?m<bornOnDate> ?bd. ?city <foundingYear> ''1718''.
   FILTER(regex(str(?bd),''1976''))
}
```

Subject	Property	Object
y:Abraham_Lincoln	hasName	"Abraham Lincoln"
y:Abraham_Lincoln	bornOnDate	"1809-02-12"
y:Abraham_Lincoln	diedOnDate	"1865-04-15"
y:Abraham_Lincoln	bornIn	y:Hodgenville_KY
y:Abraham_Lincoln	diedIn	y:Washington_DC
y:Abraham_Lincoln	title	"President"
y:Abraham_Lincoln	gender	"Male"
y:Washington_DC	hasName	"Washington D.C."
y:Washington_DC	foundingYear	"1790"
y:Hodgenville_KY	hasName	"Hodgenville"
y:United_States	hasName	"United States"
y:United_States	hasCapital	y:Washington_DC
y:United_States	foundingYear	"1776"
y:Reese_Witherspoon	bornOnDate	"1976-03-22"
y:Reese_Witherspoon	bornIn	y:New_Orleans_LA
y:Reese_Witherspoon	hasName	"Reese Witherspoon"
y:Reese_Witherspoon	gender	"Female"
y:Reese_Witherspoon	title	"Actress"
y:New_Orleans_LA	foundingYear	"1718"
y:New_Orleans_LA	locatedIn	y:United_States
y:Franklin_Roosevelt	hasName	"Franklin D. Roosevelt"
y:Franklin_Roosevelt	bornIn	y:Hyde_Park_NY
y:Franklin_Roosevelt	title	"President"
y:Franklin_Roosevelt	gender	"Male"
y:Hyde_Park_NY	foundingYear	"1810"
y:Hyde_Park_NY	locatedIn	y:United_States
y:Marilyn_Monroe	gender	"Female"
y:Marilyn_Monroe	hasName	"Marilyn Monroe"
y:Marilyn_Monroe	bornOnDate	"1926-07-01"
y:Marilyn_Monroe	diedOnDate	"1962-08-05"

问题： Too many self-joins!

```
SELECT T2.object
FROM   T as T1, T as T2, T as T3,
       T as T4
WHERE T1.property="bornIn"
AND T2.property="hasName"
AND T3.property="bornOnDate"
AND T1.subject=T2.subject
AND T2.subject=T3.subject
AND T4.propety="foundingYear"
AND T1.object=T4.subject
AND T4.object="1718"
AND T3.object LIKE '%1976%'
```

Property Tables：属性表存储

- 属性表存储也称为垂直仍然基于传统关系数据库实现，典型的如Jena [Wilkinson et al., 2003] ,FlexTable [Wang et al., 2010] , DB2-RDF [Bornea et al., 2013]等实现
- 基本思想是以实体类型为中心，把属于同一个实体类型的属性组织为一个表，即属性表进行存储。
  - 优点：Join减少了，本质上接近于关系数据库，可重用RDBMS功能
  - 缺点：很多空值，对Subject聚类比较复杂、不易处理多值属性。

```
SELECT Person.hasName
FROM Person, City
WHERE Person.bornOnDate LIKE '%1976%'
and City.foundingYear =1718
and Person.bornIn=City.Subject
```

Subject	Property	Object
y:Reese_Witherspoon	hasName	"Reese Witherspoon"
y:Reese_Witherspoon	bornOnDate	"1976-03-22"
y:Reese_Witherspoon	bornIn	y:New_Orleans_LA
y:New_Orleans_LA	hasName	"New Orleans LA"
y:New_Orleans_LA	foundingYear	"1718"
...	...	...

Subject	hasName	bornOnDate	diedOnDate	bornIn
y:Abraham_Lincoln	"Abraham Lincoln"	1809-02-12	1865-04-15	y:Hodgenville_KY
y:Reese_Witherspoon	"Reese Witherspoon"	1976-03-22		y:New_Orleans_LA

Subject	hasName	foundingYear
y:Washington_DC	"Washington D.C."	1790
y:New_Orleans_LA	"New Orleans LA"	1718

Binary Tables：二元表

- 二元表也称为垂直划分表，也是基于关系数据库实现的三元组存储方式。
- 基本思想是对三元组按属性分组，为每个属性在关系数据库中建立一个包含（subject、Object）两列的表。
- 由于一个知识图谱中属性数量是有限的，表的总体数量是可控的。
  - 优点是没有空值、也不需要聚类、对于Subject-Subject-Join操作性能好。
  - 缺点是Insert性能损耗高、并且Subject-Object Join性能差。

Subject	Property	Object
y:Abraham_Lincoln	hasName	"Abraham Lincoln"
y:Abraham_Lincoln	bornOnDate	"1809-02-12"
y:Abraham_Lincoln	diedOnDate	"1865-04-15"
y:Washington_DC	hasName	"Washington D.C."
y:Washington_DC	foundingYear	"1790"
...	...	...

hasName

Subject	Object
y:Abraham_Lincoln	"Abraham Lincoln"
y:Washington_DC	"Washington D.C."

bornOnDate

Subject	Object
y:Abraham_Lincoln	1809-02-12
y:Reese_Witherspoon	1976-03-22

foundingYear

Subject	Object
y:Washington_DC	1790
y:Hyde_Park_NY	1810

Exhaustive Indexing：全索引结构

- 性能最好的存储方式是基于全索引结构的存储，典型的实现包括RDF-3X, Hexastore等。
- 这种方法也仅维护一张包含(Subject, Predicate, Object）的三列表，但增加了多个方面的优化手段。
- 第一个优化手段是建立Mapping Table，即将所有的字符串首先映射到唯一的数字ID，这一将大大压缩存储空间。

Original triple table

Subject	Property	Object
y:Abraham_Lincoln	hasName	"Abraham Lincoln"
y:Abraham_Lincoln	bornOnDate	"1809-02-12"
y:Abraham_Lincoln	diedOnDate	"1865-04-15"
y:Washington_DC	hasName	"Washington D.C."
y:Washington_DC	foundingYear	"1790"

Encoded triple table

Subject	Property	Object
0	1	2
0	3	4
0	5	6
7	1	8
7	9	10

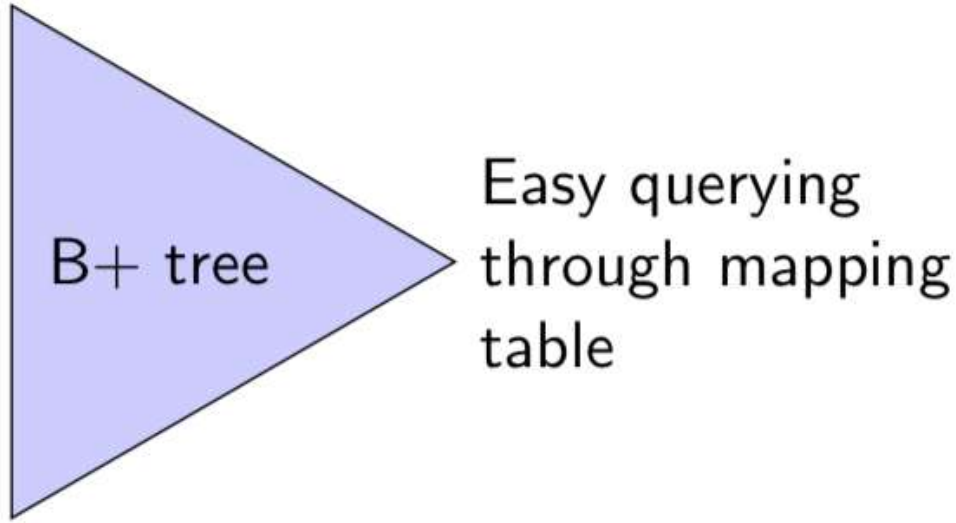
Mapping table

ID	Value
0	y:Abraham_Lincoln
1	hasName
2	"Abraham Lincoln"
3	bornOnDate
4	"1809-02-12"
5	diedOnDate
6	"1865-04-15"
7	y:Washington_DC
8	"Washington D.C."
9	foundingYear
10	"1790"

Exhaustive Indexing

- 进一步建立六种索引： SPO, SOP, PSO, POS, OPS, OSP，即分别建立Subject-Predicate-Object; Subject-Object-Predicate； Predicate-Subject-Object等六个方面的全索引，显然多种形式的索引覆盖了多个维度的图查询需求。
- 同时三元组基于字符串排序，并利用clustered B+ tree树来组织索引以进一步优化索引检索的效率。

Subject	Property	Object
0	1	2
0	3	4
0	5	6
7	1	8
7	9	10



Exhaustive Indexing查询举例

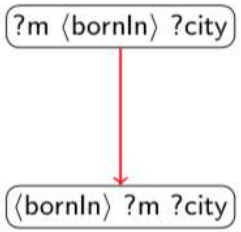
```
SELECT ?name
WHERE {
  ?m <bornIn> ?city. ?m <hasName> ?name.
  ?m<bornOnDate> ?bd. ?city <foundingYear> ''1718''.
  FILTER(regex(str(?bd),''1976''))
}
```

Subject	Property	Object
y:Abraham_Lincoln	hasName	"Abraham Lincoln"
y:Abraham_Lincoln	bornOnDate	"1809-02-12"
y:Abraham_Lincoln	diedOnDate	"1865-04-15"
y:Abraham_Lincoln	bornIn	y:Hodgenville_KY
y:Abraham_Lincoln	diedIn	y:Washington_DC
y:Abraham_Lincoln	title	"President"
y:Abraham_Lincoln	gender	"Male"
y:Washington_DC	hasName	"Washington D.C."
y:Washington_DC	foundingYear	"1790"
y:Hodgenville_KY	hasName	"Hodgenville"
y:United_States	hasName	"United States"
y:United_States	hasCapital	y:Washington_DC
y:United_States	foundingYear	"1776"
y:Reese_Witherspoon	bornOnDate	"1976-03-22"
y:Reese_Witherspoon	bornIn	y:New_Orleans_LA
y:Reese_Witherspoon	hasName	"Reese Witherspoon"
y:Reese_Witherspoon	gender	"Female"
y:Reese_Witherspoon	title	"Actress"
y:New_Orleans_LA	foundingYear	"1718"
y:New_Orleans_LA	locatedIn	y:United_States
y:Franklin_Roosevelt	hasName	"Franklin D. Roosevelt"
y:Franklin_Roosevelt	bornIn	y:Hyde_Park_NY
y:Franklin_Roosevelt	title	"President"
y:Franklin_Roosevelt	gender	"Male"
y:Hyde_Park_NY	foundingYear	"1810"
y:Hyde_Park_NY	locatedIn	y:United_States
y:Marilyn_Monroe	gender	"Female"
y:Marilyn_Monroe	hasName	"Marilyn Monroe"
y:Marilyn_Monroe	bornOnDate	"1926-07-01"
y:Marilyn_Monroe	diedOnDate	"1962-08-05"

Exhaustive Indexing查询举例

```
SELECT ?name
WHERE {
   ?m <bornIn> ?city. ?m <hasName> ?name.
   ?m<bornOnDate> ?bd. ?city <foundingYear> ''1718''.
   FILTER(regex(str(?bd),''1976''))
}
```

Subject	Property	Object
y:Abraham_Lincoln	hasName	"Abraham Lincoln"
y:Abraham_Lincoln	bornOnDate	"1809-02-12"
y:Abraham_Lincoln	diedOnDate	"1865-04-15"
y:Abraham_Lincoln	bornIn	y:Hodgenville_KY
y:Abraham_Lincoln	diedIn	y:Washington_DC
y:Abraham_Lincoln	title	"President"
y:Abraham_Lincoln	gender	"Male"
y:Washington_DC	hasName	"Washington D.C."
y:Washington_DC	foundingYear	"1790"
y:Hodgenville_KY	hasName	"Hodgenville"
y:United_States	hasName	"United States"
y:United_States	hasCapital	y:Washington_DC
y:United_States	foundingYear	"1776"
y:Reese_Witherspoon	bornOnDate	"1976-03-22"
y:Reese_Witherspoon	bornIn	y:New_Orleans_LA
y:Reese_Witherspoon	hasName	"Reese Witherspoon"
y:Reese_Witherspoon	gender	"Female"
y:Reese_Witherspoon	title	"Actress"
y:New_Orleans_LA	foundingYear	"1718"
y:New_Orleans_LA	locatedIn	y:United_States
y:Franklin_Roosevelt	hasName	"Franklin D. Roosevelt"
y:Franklin_Roosevelt	bornIn	y:Hyde_Park_NY
y:Franklin_Roosevelt	title	"President"
y:Franklin_Roosevelt	gender	"Male"
y:Hyde_Park_NY	foundingYear	"1810"
y:Hyde_Park_NY	locatedIn	y:United_States
y:Marilyn_Monroe	gender	"Female"
y:Marilyn_Monroe	hasName	"Marilyn Monroe"
y:Marilyn_Monroe	bornOnDate	"1926-07-01"
y:Marilyn_Monroe	diedOnDate	"1962-08-05"



sorted by PSO

Property	Subject	Object
bornIn	y:Franklin_Roosevelt	y:Hyde_Park_NY
bornIn	y:Reese_Witherspoon	y:New_Orleans_LA
bornOnDate	y:Abraham_Lincoln	"1809-02-12"
bornOnDate	y:Marilyn_Monroe	"1926-07-01"
bornOnDate	y:Reese_Witherspoon	"1976-03-22"
diedIn	y:Abraham_Lincoln	y:Washington_DC
diedOnDate	y:Abraham_Lincoln	"1865-04-15"
diedOnDate	y:Marilyn_Monroe	"1962-08-05"
foundingYear	y:Hyde_Park_NY	"1810"
foundingYear	y:New_Orleans_LA	"1718"
foundingYear	y:United_States	"1776"
foundingYear	y:Washington_DC	"1790"
gender	y:Abraham_Lincoln	"Male"
gender	y:Franklin_Roosevelt	"Male"
gender	y:Marilyn_Monroe	"Female"
gender	y:Reese_Witherspoon	"Female"
hasCapital	y:United_States	y:Washington_DC
hasName	y:Abraham_Lincoln	"Abraham Lincoln"
hasName	y:Franklin_Roosevelt	"Franklin D. Roosevelt"
hasName	y:Hodgenville_KY	"Hodgenville"
hasName	y:Marilyn_Monroe	"Marilyn Monroe"
hasName	y:Reese_Witherspoon	"Reese Witherspoon"
hasName	y:United_States	"United States"
hasName	y:Washington_DC	"Washington D.C."
locatedIn	y:Hyde_Park_NY	y:United_States
locatedIn	y:New_Orleans_LA	y:United_States
title	y:Abraham_Lincoln	"President"
title	y:Franklin_Roosevelt	"President"
title	y:Reese_Witherspoon	"Actress"

Exhaustive Indexing查询举例

```
SELECT ?name
WHERE {
  ?m <bornIn> ?city. ?m <hasName> ?name.
  ?m<bornOnDate> ?bd. ?city <foundingYear> ''1718''.
  FILTER(regex(str(?bd),''1976''))
}
```

Subject	Property	Object
y:Abraham_Lincoln	hasName	"Abraham Lincoln"
y:Abraham_Lincoln	bornOnDate	"1809-02-12"
y:Abraham_Lincoln	diedOnDate	"1865-04-15"
y:Abraham_Lincoln	bornIn	y:Hodgenville_KY
y:Abraham_Lincoln	diedIn	y:Washington_DC
y:Abraham_Lincoln	title	"President"
y:Abraham_Lincoln	gender	"Male"
y:Washington_DC	hasName	"Washington D.C."
y:Washington_DC	foundingYear	"1790"
y:Hodgenville_KY	hasName	"Hodgenville"
y:United_States	hasName	"United States"
y:United_States	hasCapital	y:Washington_DC
y:United_States	foundingYear	"1776"
y:Reese_Witherspoon	bornOnDate	"1976-03-22"
y:Reese_Witherspoon	bornIn	y:New_Orleans_LA
y:Reese_Witherspoon	hasName	"Reese Witherspoon"
y:Reese_Witherspoon	gender	"Female"
y:Reese_Witherspoon	title	"Actress"
y:New_Orleans_LA	foundingYear	"1718"
y:New_Orleans_LA	locatedIn	y:United_States
y:Franklin_Roosevelt	hasName	"Franklin D. Roosevelt"
y:Franklin_Roosevelt	bornIn	y:Hyde_Park_NY
y:Franklin_Roosevelt	title	"President"
y:Franklin_Roosevelt	gender	"Male"
y:Hyde_Park_NY	foundingYear	"1810"
y:Hyde_Park_NY	locatedIn	y:United_States
y:Marilyn_Monroe	gender	"Female"
y:Marilyn_Monroe	hasName	"Marilyn Monroe"
y:Marilyn_Monroe	bornOnDate	"1926-07-01"
y:Marilyn_Monroe	diedOnDate	"1962-08-05"

?m ⟨hasName⟩ ?name

sorted by PSO

Property	Subject	Object
bornIn	y:Franklin_Roosevelt	y:Hyde_Park_NY
bornIn	y:Reese_Witherspoon	y:New_Orleans_LA
bornOnDate	y:Abraham_Lincoln	"1809-02-12"
bornOnDate	y:Marilyn_Monroe	"1926-07-01"
bornOnDate	y:Reese_Witherspoon	"1976-03-22"
diedIn	y:Abraham_Lincoln	y:Washington_DC
diedOnDate	y:Abraham_Lincoln	"1865-04-15"
diedOnDate	y:Marilyn_Monroe	"1962-08-05"
foundingYear	y:Hyde_Park_NY	"1810"
foundingYear	y:New_Orleans_LA	"1718"
foundingYear	y:United_States	"1776"
foundingYear	y:Washington_DC	"1790"
gender	y:Abraham_Lincoln	"Male"
gender	y:Franklin_Roosevelt	"Male"
gender	y:Marilyn_Monroe	"Female"
gender	y:Reese_Witherspoon	"Female"
hasCapital	y:United_States	y:Washington_DC
hasName	y:Abraham_Lincoln	"Abraham Lincoln"
hasName	y:Franklin_Roosevelt	"Franklin D. Roosevelt"
hasName	y:Hodgenville_KY	"Hodgenville"
hasName	y:Marilyn_Monroe	"Marilyn Monroe"
hasName	y:Reese_Witherspoon	"Reese Witherspoon"
hasName	y:United_States	"United States"
hasName	y:Washington_DC	"Washington D.C."
locatedIn	y:Hyde_Park_NY	y:United_States
locatedIn	y:New_Orleans_LA	y:United_States
title	y:Abraham_Lincoln	"President"
title	y:Franklin_Roosevelt	"President"
title	y:Reese_Witherspoon	"Actress"

小结

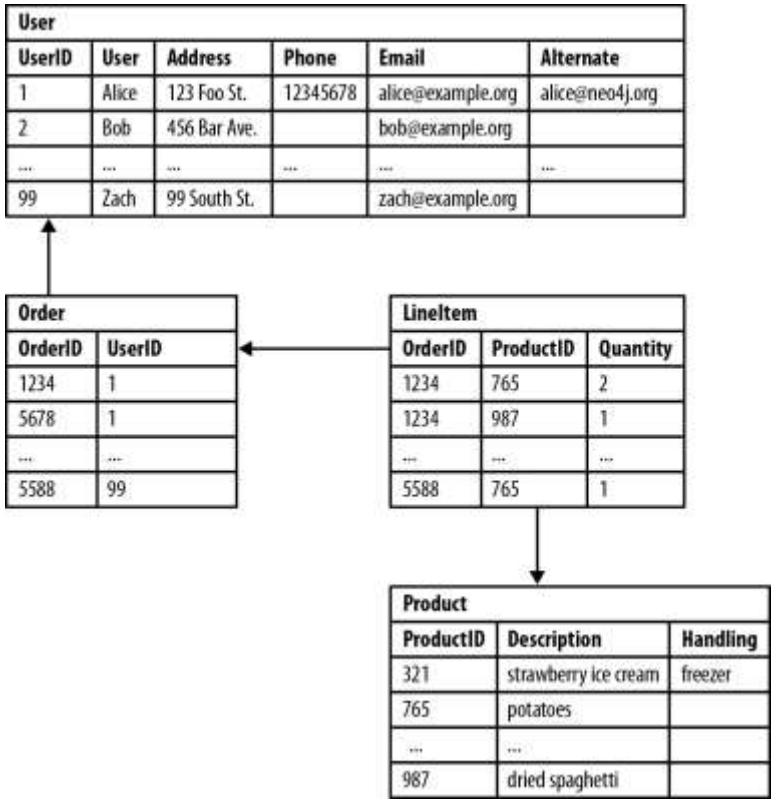
存储方案		优点	缺点	代表性系统	文献
基于关系	三元组表	① 存储结构简单	① 大量自连接操作开销巨大	3store	[53]
	水平表	① 知识图谱的邻接表,存储方案简单	① 可能超出所允许的表中列数目的上限 ② 表中可能存在大量空值 ③ 无法表示一对多联系或多值属性 ④ 谓语的增加、修改或删除成本高	DLDB	[54]
	属性表	① 克服了三元组表的自连接问题 ② 解决了水平表中列数目过多的问题	① 真实知识图谱需建立的关系表数量可能超过上限 ② 由于知识图谱的灵活性,表中可能存在大量空值 ③ 无法表示一对多联系或多值属性	Jena	[55]
	垂直划分	① 解决了空值问题 ② 解决了多值问题 ③ 能够快速执行不同谓语表的连接查询	① 真实知识图谱需维护大量谓语表 ② 复杂知识图谱查询需执行的表连接操作 ③ 数据更新维护代价大	SW-Store	[56]
	六重索引	① 每种三元组模式查询均可直接使用对应索引快速查找 ② 通过不同索引表之间的连接操作直接加速知识图谱上的连接查询	① 需要花费6倍的存储空间开销和数据更新维护代价 ② 复杂知识图谱查询会产生大量索引表连接查询操作	RDF-3X Hexastore	[58,59]

from《知识图谱数据管理研究综述》王鑫等

第3节 基于原生图数据库的图存储

关系型数据库的局限性：关系模型不善处理“关系”

- 关系模型将语义关联关系隐藏在外键结构中，无显示表达，并带来关联查询与计算的复杂性。
- 数据来源多样性带来大量离群数据（Outlier Data），导致数据集的宏观结构愈发复杂和不规整，对于包含大量离群数据的场景，关系模型将造成大量表连接、稀疏行和非空处理。
- 互联网的开放世界假设要求数据模型满足高动态和去中心化的扩增数据的能力，关系模型对表结构的范式要求限制了Schema层的动态性。



User

UserID	User	Address	Phone	Email	Alternate
1	Alice	123 Foo St.	12345678	alice@example.org	alice@neo4j.org
2	Bob	456 Bar Ave.		bob@example.org	
...	...	...	...	...	...
99	Zach	99 South St.		zach@example.org	

Order

OrderID	UserID
1234	1
5678	1
...	...
5588	99

LineItem

OrderID	ProductID	Quantity
1234	765	2
1234	987	1
...	...	...
5588	765	1

Product

ProductID	Description	Handling
321	strawberry ice cream	freezer
765	potatoes	
...	...	
987	dried spaghetti	

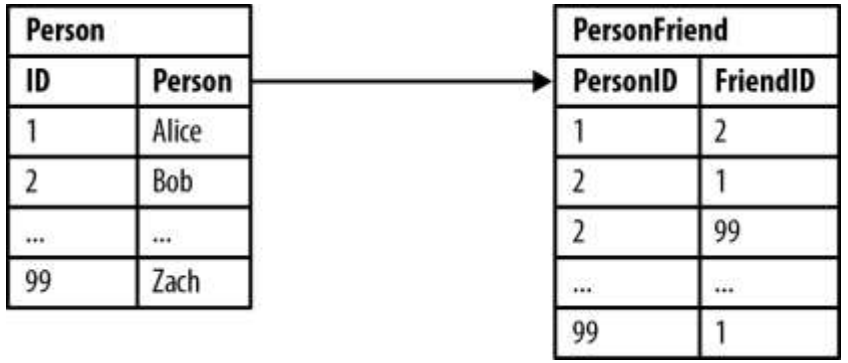
关系模型背离了用接近自然语言的方式来描述客观世界的原则，这使得概念化、高度关联的世界模型与数据的物理存储之间出现了失配。

关系模型的局限性：处理关联查询

1. Asking “who are Bob’s friends?” is easy

Example 2-1. Bob’s friends

```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
  ON PersonFriend.FriendID = p1.ID
JOIN Person p2
  ON PersonFriend.PersonID = p2.ID
WHERE p2.Person = 'Bob'
```

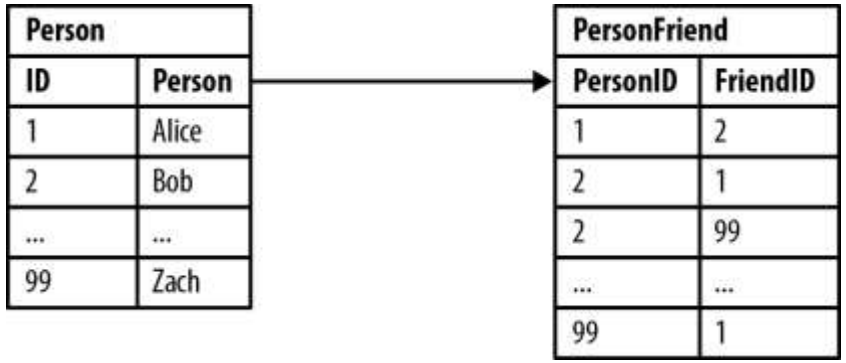


Person	
ID	Person
1	Alice
2	Bob
...	...
99	Zach

PersonFriend	
PersonID	FriendID
1	2
2	1
2	99
...	...
99	1

关系模型的局限性：处理关联查询

2. 处理Reflexive relationship的自反查询带来底层数据库的更多计算，从图的角度，只需声明friend关系是自反关系即可支持反向查询。



Person	
ID	Person
1	Alice
2	Bob
...	...
99	Zach

PersonFriend	
PersonID	FriendID
1	2
2	1
2	99
...	...
99	1

Example 2-1. Bob's friends

```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
  ON PersonFriend.FriendID = p1.ID
JOIN Person p2
  ON PersonFriend.PersonID = p2.ID
WHERE p2.Person = 'Bob'
```

Example 2-2. Who is friends with Bob?

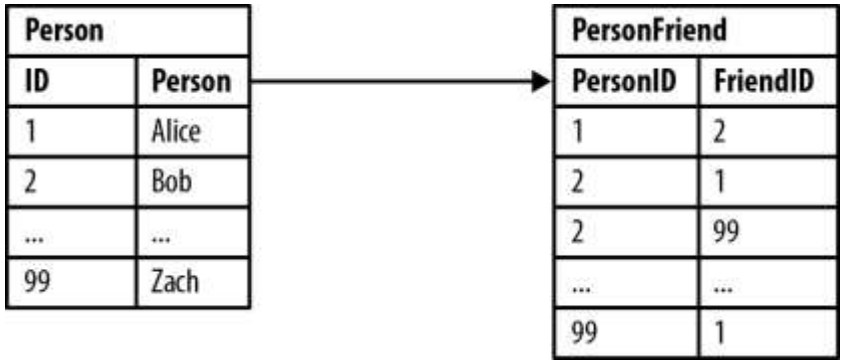
```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
  ON PersonFriend.PersonID = p1.ID
JOIN Person p2
  ON PersonFriend.FriendID = p2.ID
WHERE p2.Person = 'Bob'
```

关系模型的局限性：处理关联查询

3. 处理Chain relationship 的多跳查询带来更加复杂的Join计算。有些数据库如Oracle提供了Syntax Suger如“Connect By”，但没有降低底层的计算复杂度。

Example 2-3. Alice's friends-of-friends

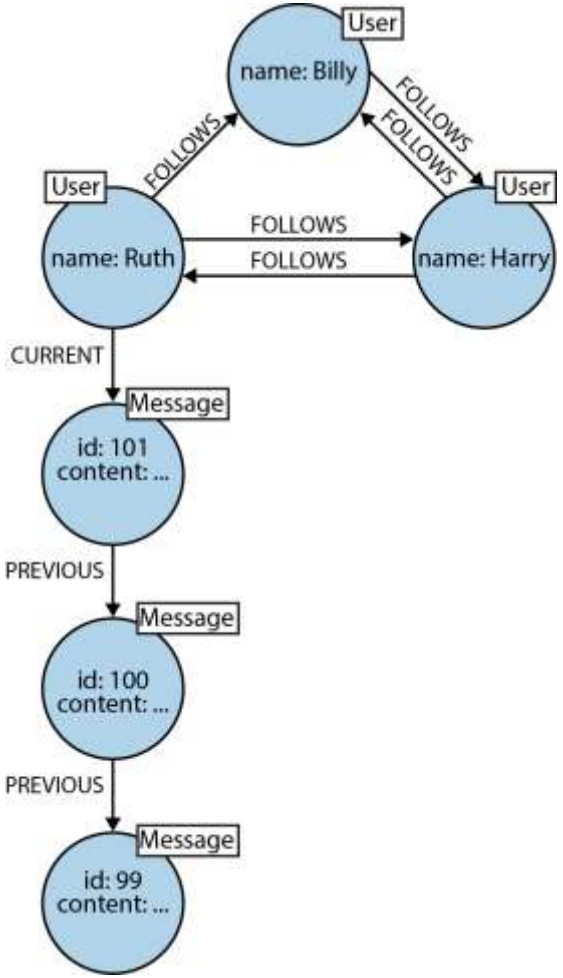
```
SELECT p1.Person AS PERSON, p2.Person AS FRIEND_OF_FRIEND
FROM PersonFriend pf1 JOIN Person p1
  ON pf1.PersonID = p1.ID
JOIN PersonFriend pf2
  ON pf2.PersonID = pf1.FriendID
JOIN Person p2
  ON pf2.FriendID = p2.ID
WHERE p1.Person = 'Alice' AND pf2.FriendID <> p1.ID
```



Person	
ID	Person
1	Alice
2	Bob
...	...
99	Zach

PersonFriend	
PersonID	FriendID
1	2
2	1
2	99
...	...
99	1

关系数据库的局限性：处理关联查询



举例一：找出两个银行账户交易记录的最短路径
举例二：找出两个社交账户消息发送的最短路径

客户信息表

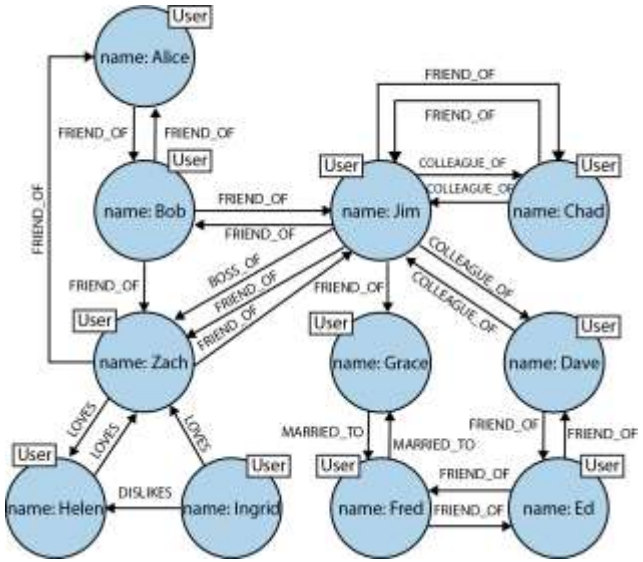
User_ID	User_Name	…
1	A	
2	B	
…		
…	C	

交易信息表 or 消息发送表

from_ID	to_id	…
1	3	
3	11	
…	…	
315	325	

关系数据库的局限性：处理关联查询

1,000,000 people, each with approximately 50 friends, query to a maximum depth of five.



Depth	RDBMS execution time(s)	Neo4j execution time(s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

关系数据库的局限性：

知识图谱需要更加丰富的关系语义表达与关联推理能力

- 在需要更加深入的研究数据之间的关系时，需要更加丰富的关系语义的表达能力，除了前述Reflesive和多跳关系、还包括传递关系Transitive、对称关系（非对称关系）Symmetric、反关系（Inverse）、函数关系（Functional）。
- 除了关联查询能力，深层次的关系建模还将提供关联推理的能力，属性图数据库如Neo4J提供了由于关系模型的关联查询能力，AllegroGraph等RDF图数据库提供了更多的关联推理能力。

Symmetric Property

```
:hasSpouse  rdf:type  owl:SymmetricProperty .
```

Transitive Property

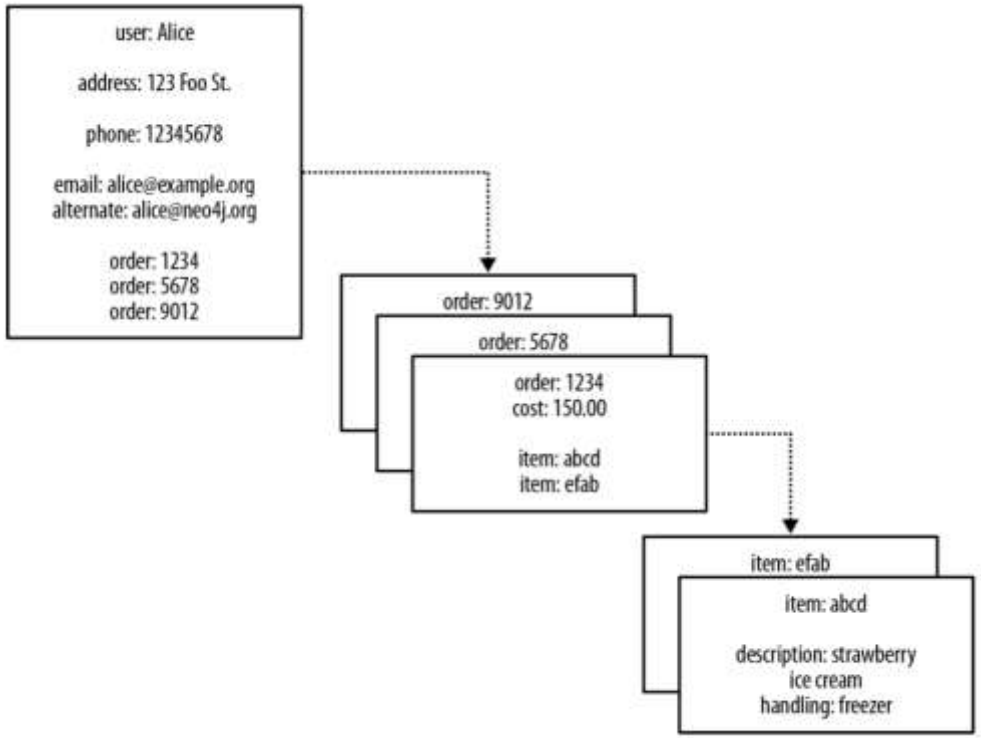
```
:hasAncestor  rdf:type  owl:TransitiveProperty .
```

Property Chain

```
:hasGrandparent  owl:propertyChainAxiom  ( :hasParent  :hasParent ) .
```

NoSQL数据库也不善于处理关联关系

- 关系在NoSQL数据库中也不是First-Class Citizen，在处理数据关联也需要使用类似于外键的Foreign Aggregates。
- Foreign Aggregates不能处理自反关系，例如，查询“who is friends with Bob?”时，需要暴力计算，即扫描所有实体数据集。
- Foreign Aggregates也不负责维护Link的有效性，在处理多跳关系时效率也很低下。



图数据库： Relations are first-class citizens

关系模型中关系被隐藏定义

Company Table

ID	Company Name
IDC	Tencent

City Table

City	Country
Shengzhen	China

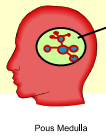
Company_City Table

City_ID	CO_ID
Shengzhen	IDC

Where are the relationships?

Relationships are in documents, SQL code and collective memories - not available to applications!

Data Definition Statements? Applications do not use them, they are not descriptive and their scope is a single database



Data Dictionary? Data Registry? They are for human, not computer use

Question

In which country is Tencent located?

Develop a Query

Select Country

From Company Table, City Table, Company_City Table

Where Company Name = "Tencent" and ID = CO_ID and City = City_ID

Answer

In China

图数据模型中关系被显示描述

Information Given

Tencent *located_in* Shengzhen

Shengzhen *located_in* China

Where are the relationships?

Relationship Model

located_in type transitiveProperty

Information Inferred

Tencent *located_in* China

Relationships are explicit in the model and directly available to applications!

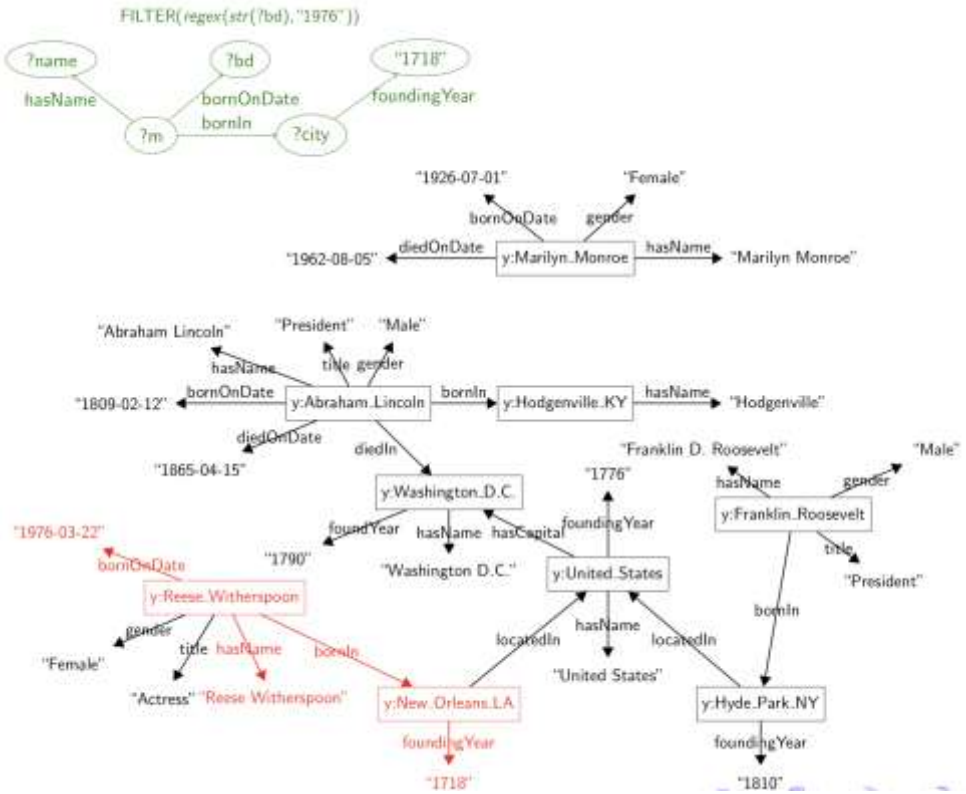
Question

In which country is Tencent located?

Answer

In China

原生图数据库：利用图的结构特征建索引



Represent G as Adjacency List

Prefix: y=http://en.wikipedia.org/wiki/

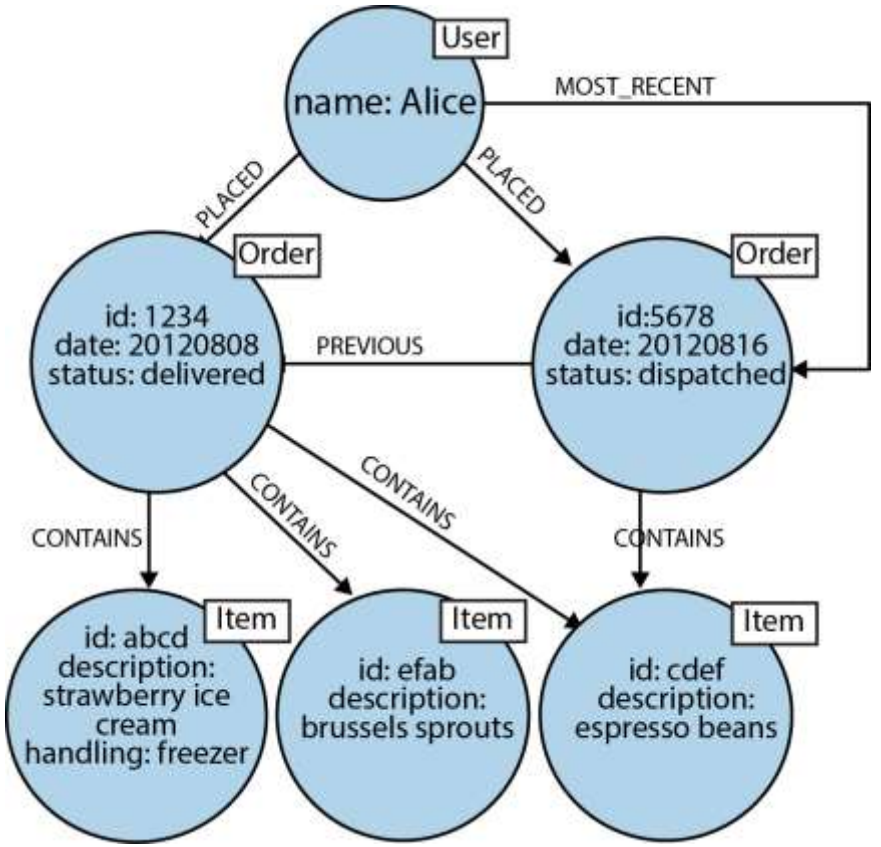
Label	adjList
⋮	⋮
y:Reese_Witherspoon	(BornOnDate, "1976-03-22"), (gender, "Female"), (title, "Actress"), (hasName, "Reese Witherspoon"), (BornIn,New_Orleans_LA)
⋮	⋮

0010 1000

Encode all neighbors of a vertex into a bit string vertex signature

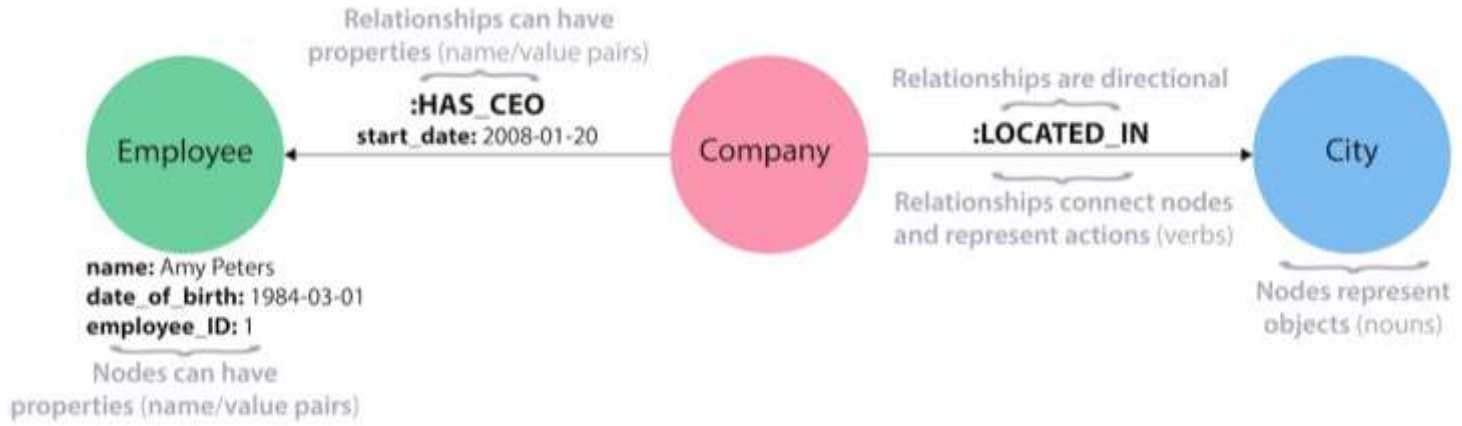
图数据建模的好处

- 自然表达：图是十分自然的描述事物关系的方式，更加接近于人脑对客观事物的记忆方式。
- 易于扩展：图模型更加易于适应变化，例如在图中，临时希望获取历史订单，只需新增边即可。
- 复杂关联表达：图模型易于表达复杂关联逻辑的查询，例如在推荐系统中，希望表达复杂的推荐逻辑，例如：“all the flavors of ice cream liked by people who enjoy espresso but dislike Brussels sprouts, and who live in a particular neighborhood.”
- 多跳优化：在处理多跳查询上，图模型有性能优势。

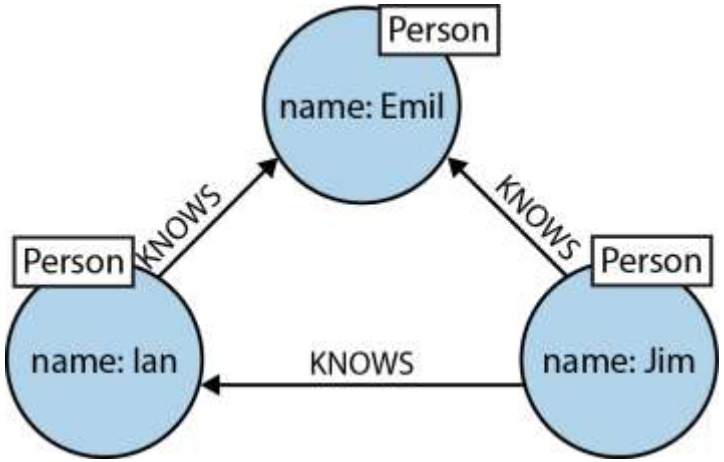


图数据库使用举例：Neo4J

- 属性图是图数据库Neo4J实现的图结构表示模型，在工业界有广泛应用。
- 在属性图的术语中，属性图是由 顶点（Vertex），边（Edge），标签（Label），关系类型还有属性（Property）组成的有向图。
- 在属性图中，节点和关系是最重要的实体。节点上包含属性，属性可以以任何键值形式存在。



图查询语言: Cypher



Cypher of Neo4J

```
MATCH (a:Person {name:'Jim'})-
[:KNOWS]->(b)-[:KNOWS]->(c),
      (a)-[:KNOWS]->(c)
RETURN b, c
```

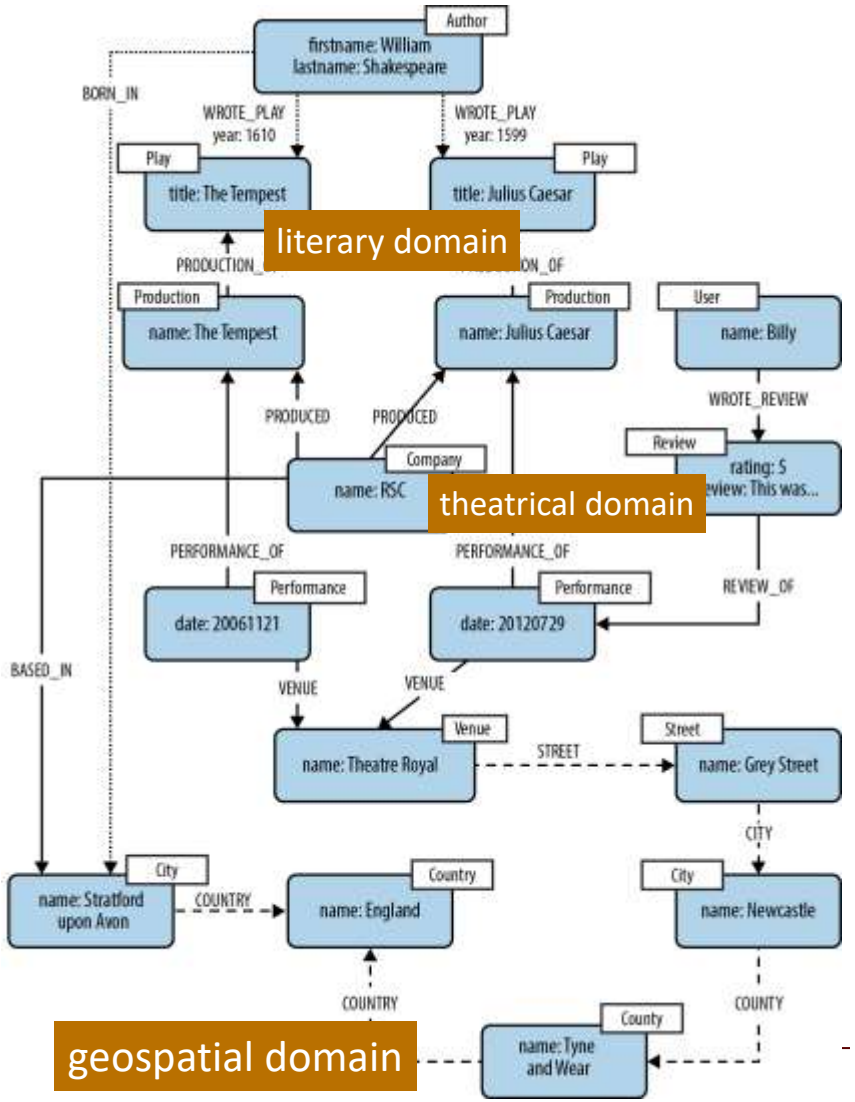
SPARQL of W3C

```
Select ?a,?b,?c where {
  ?a :KNOWS ?b.?b :KNOWS ?c.
  ?a :KNOWS ?c.
  ?a ISA :Person.
  ?a :name = "Jim".
}
```

Gremlin of Apache TinkerPop

```
g.V().has("name","Jim").
  out(":KNOWS").
  out(":KNOWS").
  values("name")
```

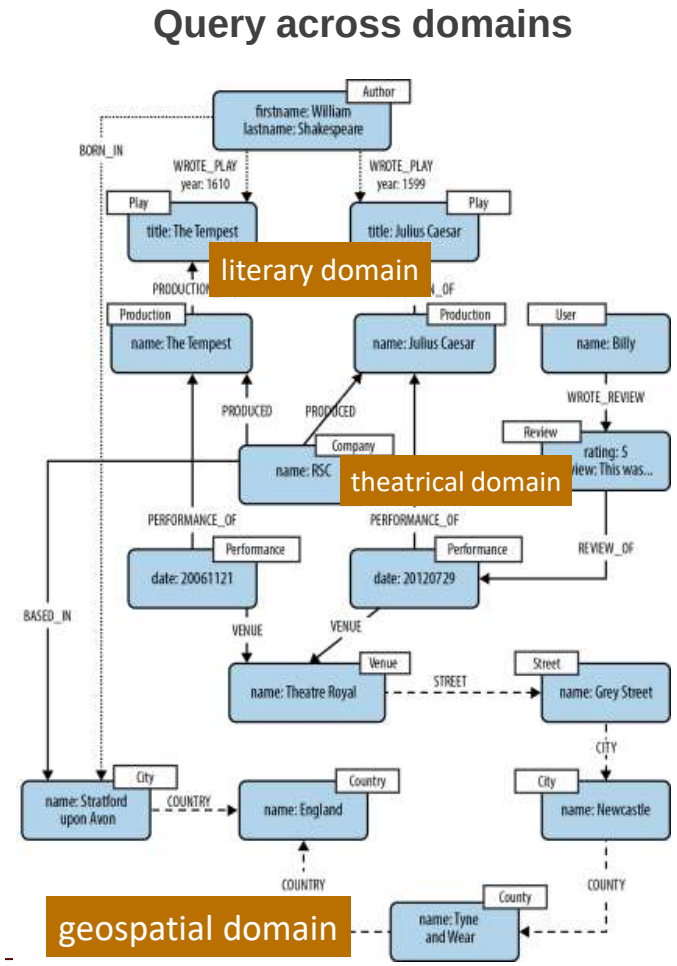
举例： 跨领域图建模与查询



- 利用图谱将不同领域的数据进行关联
- 要求模型能按需扩展
- 查询语句能表示跨多个领域的关联逻辑

```
CREATE (shakespeare:Author {firstname:'William', lastname:'Shakespeare'}),
       (juliusCaesar:Play {title:'Julius Caesar'}),
       (shakespeare)-[:WROTE_PLAY {year:1599}]->(juliusCaesar),
       (theTempest:Play {title:'The Tempest'}),
       (shakespeare)-[:WROTE_PLAY {year:1610}]->(theTempest),
       (rsc:Company {name:'RSC'}),
       (production1:Production {name:'Julius Caesar'}),
       (rsc)-[:PRODUCED]->(production1),
       (production1)-[:PRODUCTION_OF]->(juliusCaesar),
       (performance1:Performance {date:20120729}),
       (performance1)-[:PERFORMANCE_OF]->(production1),
       (production2:Production {name:'The Tempest'}),
       (rsc)-[:PRODUCED]->(production2),
       (production2)-[:PRODUCTION_OF]->(theTempest),
       (performance2:Performance {date:20061121}),
       (performance2)-[:PERFORMANCE_OF]->(production2),
       (performance3:Performance {date:20120730}),
       (performance3)-[:PERFORMANCE_OF]->(production1),
       (billy:User {name:'Billy'}),
       (review:Review {rating:5, review:'This was awesome!'}),
       (billy)-[:WROTE_REVIEW]->(review),
       (review)-[:RATED]->(performance1),
       (theatreRoyal:Venue {name:'Theatre Royal'}),
       (performance1)-[:VENUE]->(theatreRoyal),
       (performance2)-[:VENUE]->(theatreRoyal),
       (performance3)-[:VENUE]->(theatreRoyal),
       (greyStreet:Street {name:'Grey Street'}),
       (theatreRoyal)-[:STREET]->(greyStreet),
       (newcastle:City {name:'Newcastle'}),
       (greyStreet)-[:CITY]->(newcastle),
       (tyneAndWear:County {name:'Tyne and Wear'}),
       (newcastle)-[:COUNTY]->(tyneAndWear),
       (england:Country {name:'England'}),
       (tyneAndWear)-[:COUNTRY]->(england),
       (stratford:City {name:'Stratford upon Avon'}),
       (stratford)-[:COUNTRY]->(england),
       (rsc)-[:BASED_IN]->(stratford),
       (shakespeare)-[:BORN_IN]->stratford
```

Cypher图查询举例

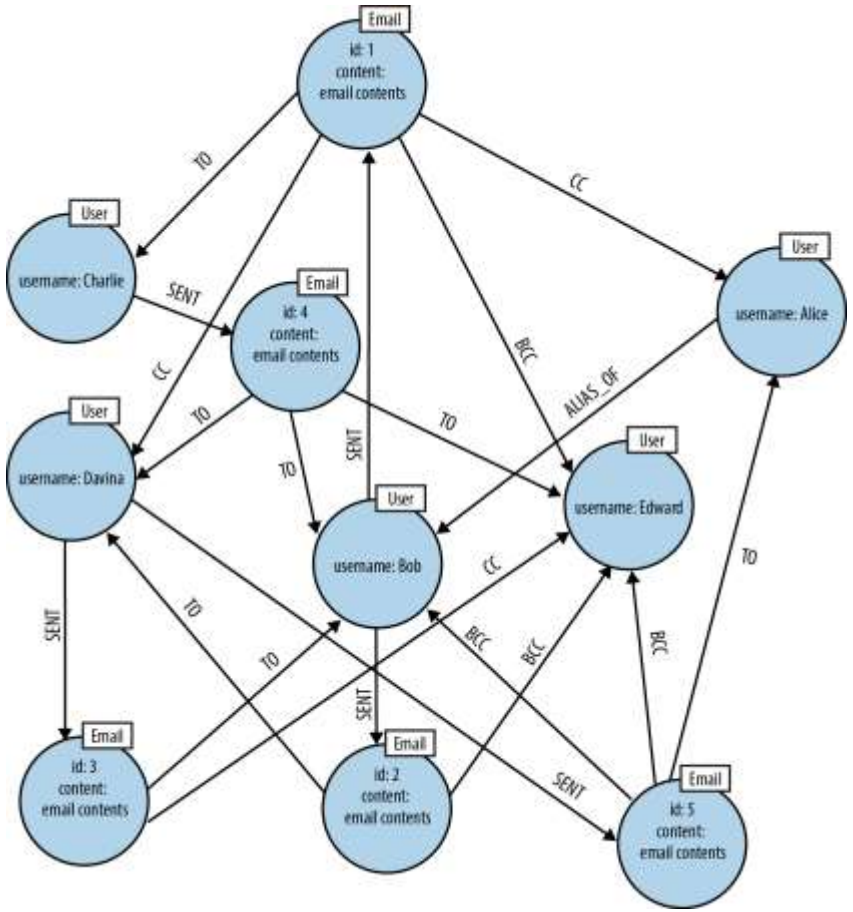


Finds all the Shakespeare performances at Newcastle's Theatre Royal

```
MATCH (theater:Venue {name:'Theatre Royal'}),
      (newcastle:City {name:'Newcastle'}),
      (bard:Author {lastname:'Shakespeare'}),
      (newcastle)<-[:STREET|CITY*1..2]-(theater)
        <-[:VENUE]-()-[:PERFORMANCE_OF]->()
        -[:PRODUCTION_OF]->(play)<-[w:WROTE_PLAY]-(bard)
WHERE w.year > 1608
RETURN DISTINCT play.title AS play
```

```
+-----------------+
| play            |
+-----------------+
| "Julius Caesar" |
| "The Tempest"   |
+-----------------+
2 rows
```

Cypher图查询举例



Identify potentially suspect behavior:retrieve all the emails that Bob has sent where he's CC'd one of his own aliases

```
MATCH (bob:User {username:'Bob'})-
[:SENT]->(email)-[:CC]->(alias),
      (alias)-[:ALIAS_OF]->(bob)
RETURN email.id
```

```
+-----------------------------------------+
| email                                   |
+-----------------------------------------+
| Node[6]{id:"1",content:"email contents"} |
+-----------------------------------------+
1 row
```

常见的图数据库列表

名称	License	Language	简介
AllegroGraph	Proprietary, clients: Eclipse Public License v1	C#, C, Common Lisp, Java, Python	RDF原生图存储，支持SPARQL，支持部分OWL推理，支持Prolog规则推理，有可视化工具
Amazon Neptune	Proprietary	Not disclosed	亚马逊的图数据库云服务，支持属性图和RDF，属性图存储支持Apache ThinkerPop Gremlin，RDF存储支持SPARQL
AnzoGraph	Proprietary	C, C++	Cambridge Semantics提供的原生图数据库引擎，支持交互式的RDF数据分析，底层支持与HDFS等大数据引擎对接，支持W3C SPARQL语言
ArangoDB	Free Apache 2, Proprietary,部分开源	C++, JavaScript, Java, Python, Node.js, PHP, Scala, Go, Ruby	基于NoSQL实现的图数据库，支持ArangoDB自己的图查询语言AQL，适合那些综合需要NoSQL存储和图存储的场景。
DataStaxEnterprise Graph	Proprietary	Java	分布式实时可扩展图数据库，底层与Cassandra集成，支持ThinkerPop Gremlin查询语言。
JanusGraph	Apache 2开源	Java	Linux Foundation下的开源分布式图数据库，支持与各种大数据存储后端集成，如Cassandra，HBase等，支持与Giraph等图计算引擎集成，支持ThinkerPop Gremlin查询语言。
MarkLogic	Proprietary, freeware developer version	Java	基于NoSQL实现的图数据库，支持RDF和SPARQL查询。
Neo4j	GPLv3 Community Edition, commercial& AGPLv3 options for enterprise	Java, .NET, JavaScript, Python, Ruby	原生图数据库，支持属性图和自定义的Cypher查询语言。
OpenLink Virtuoso	Open Source Edition is GPLv2, Enterprise Edition is proprietary	C, C++	老牌的RDF图数据库，支持RDF，SPARQL。底层基于关系数据库实现。
OrientDB	Community Edition is Apache 2, Enterprise Edition is commercial	Java	基于NoSQL实现的图数据库，支持属性图
Giraph			支持RDF图存储，支持并行图计算
BlazeGraph	GPLv2 or commercial	Java	Wikidata的后端，支持RDF推理，支持SPARQL，支持基于RDF的图挖掘与分析

小结： 什么时候使用图数据库

高性能关系查询

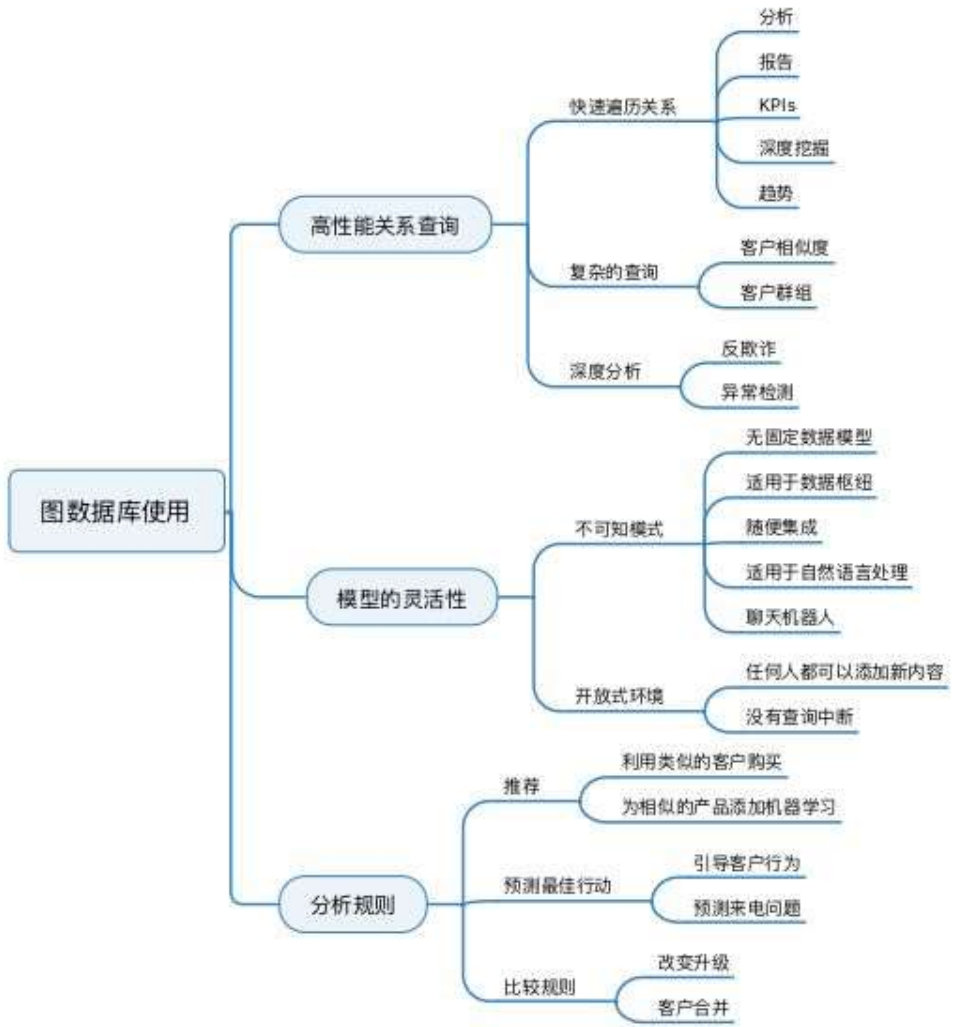
需要快速遍历许多复杂关系的任何用例。这实际上包括欺诈检测，社交网络分析，网络和数据库基础设施等。

模型的灵活性

任何依赖于添加新数据而不会中断现有查询池的用例。模型灵活性包括链接元数据，版本控制数据和不断添加新关系。

快速和复杂的分析规则

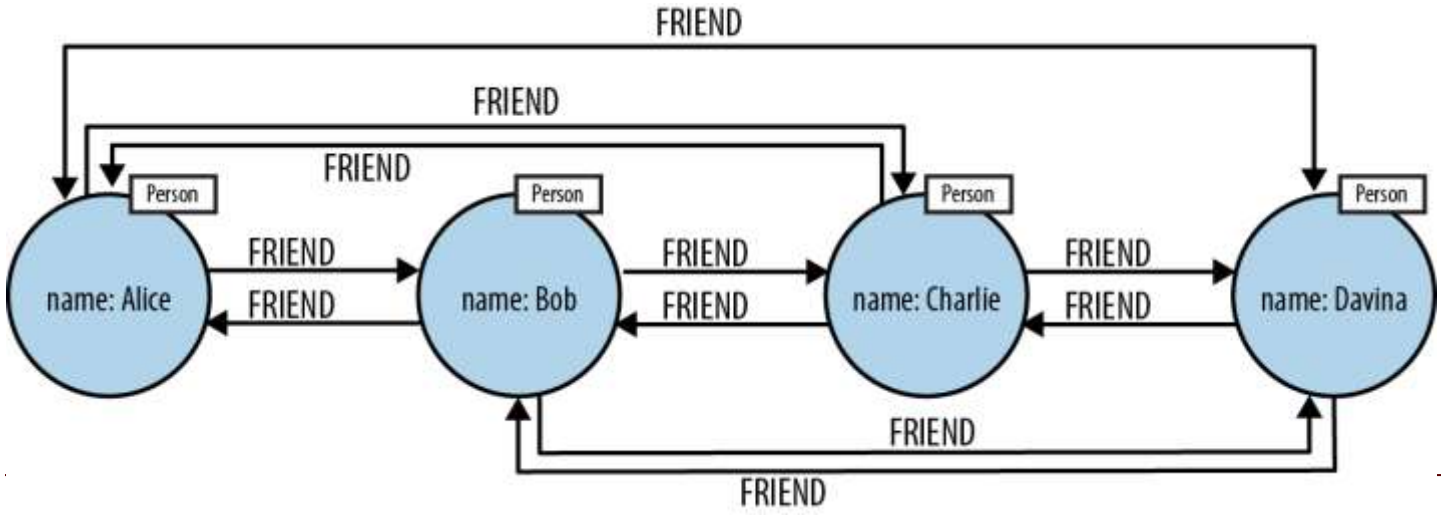
当必须执行许多复杂的规则时，例如子图的比较。这包括推荐，相似度计算和主数据管理。



第4节 原生图数据存储原理解析

原生图数据库的实现原理：免索引邻接

- 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）。
- 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。



常见Table Join的计算复杂度

Nested Join O(M*N)

```
FOR erow IN (select * from employees where X=Y) LOOP
  FOR drow IN (select * from departments where erow is matched) LOOP
    output values from erow and drow
  END LOOP
END LOOP
```

Hash Join O(M + N)

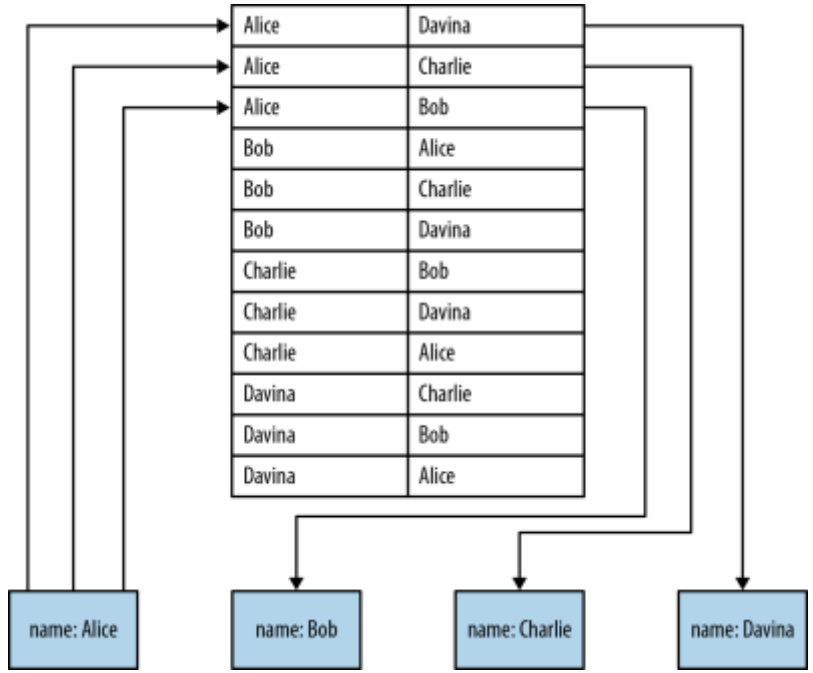
```
FOR small_table_row IN (SELECT * FROM small_table)
LOOP
  slot_number := HASH(small_table_row.join_key);
  INSERT_HASH_TABLE(slot_number,small_table_row);
END LOOP;
```

```
FOR large_table_row IN (SELECT * FROM large_table)
LOOP
   slot_number := HASH(large_table_row.join_key);
   small_table_row = LOOKUP_HASH_TABLE(slot_number,large_table_row.join_key);
   IF small_table_row FOUND
   THEN
      output small_table_row + large_table_row;
   END IF;
END LOOP;
```

Merge Join O(N*Log(N) + M*Log(M))

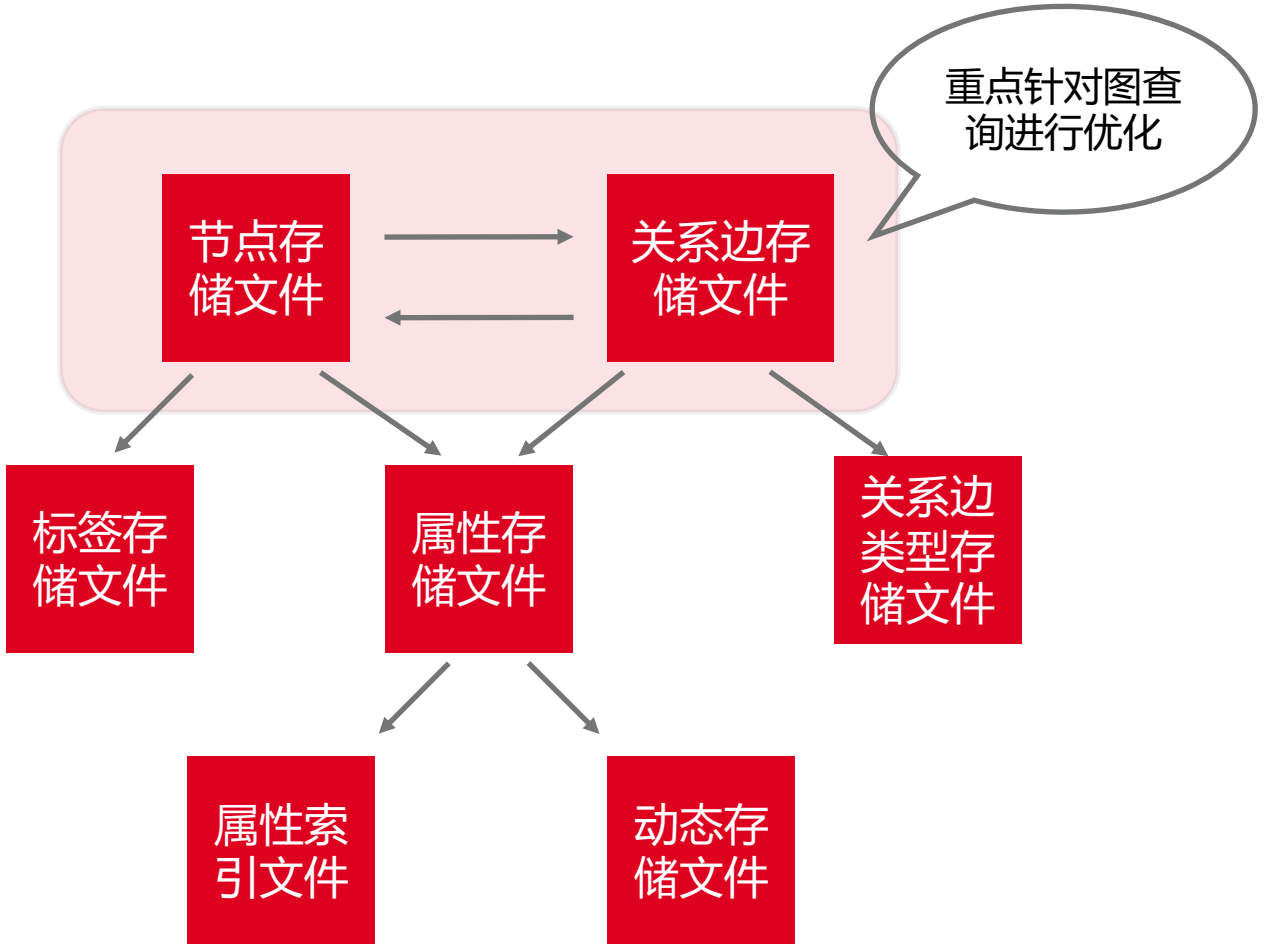
```
READ data_set_1 SORT BY JOIN KEY TO temp_ds1
READ data_set_2 SORT BY JOIN KEY TO temp_ds2

READ ds1_row FROM temp_ds1
READ ds2_row FROM temp_ds2

WHILE NOT eof ON temp_ds1,temp_ds2
LOOP
    IF ( temp_ds1.key = temp_ds2.key ) OUTPUT JOIN ds1_row,ds2_row
    ELSIF ( temp_ds1.key <= temp_ds2.key ) READ ds1_row FROM temp_ds1
    ELSIF ( temp_ds1.key => temp_ds2.key ) READ ds2_row FROM temp_ds2
END LOOP
```

Index-free adjacency的计算复杂度

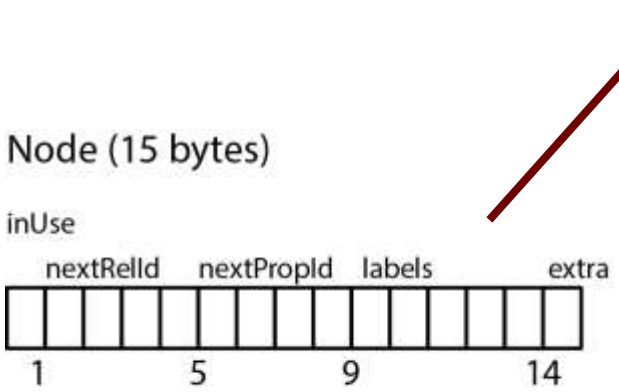


- 而对于 Index-free adjacency，关系是直接基于某个节点的相邻节点获取的(tail to head, or head to tail)；
- 例如，为了查询“who is friends with Alice”，我们只需要检索 Alice 的所有 incoming FRIEND 关系即可；
- 这个复杂度仅与节点的邻居个数有关，而与整个数据集的大小是无关的。

原生图数据库的物理存储实现

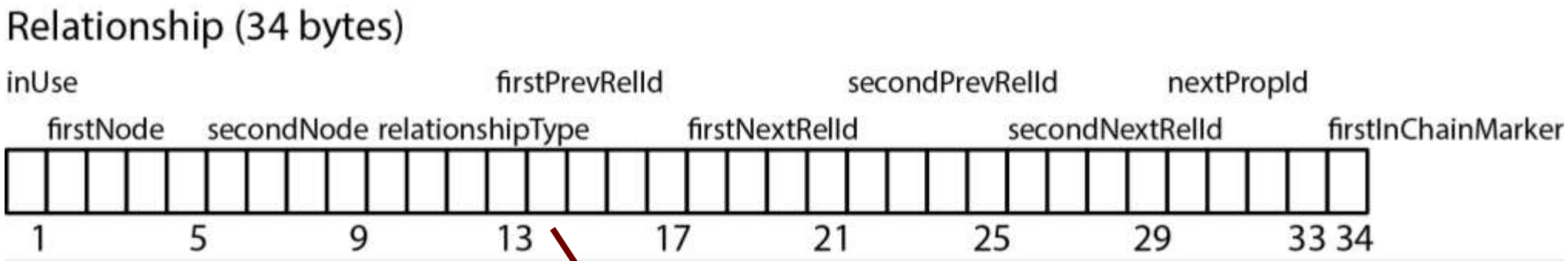


节点存储文件



- 节点存储于独立的“节点存储文件”，每个节点的存储空间固定，如14字节，便于直接通过ID编号计算获得访问地址，基于这种格式，节点查询成本为O(1)，而非O(N)
- 每个节点首字节标明是否inUse，接下来四个字节存储该节点的第一个关系边的ID，再接下来四个字节存储该节点的第一个属性ID，再接下来4个字节存储该节点的第一个Label ID。
- 节点的属性数据（如姓名、年龄等）是分开存储的，节点只存储其第一个属性的ID，这样的设计是为了保证节点遍历的高效性。

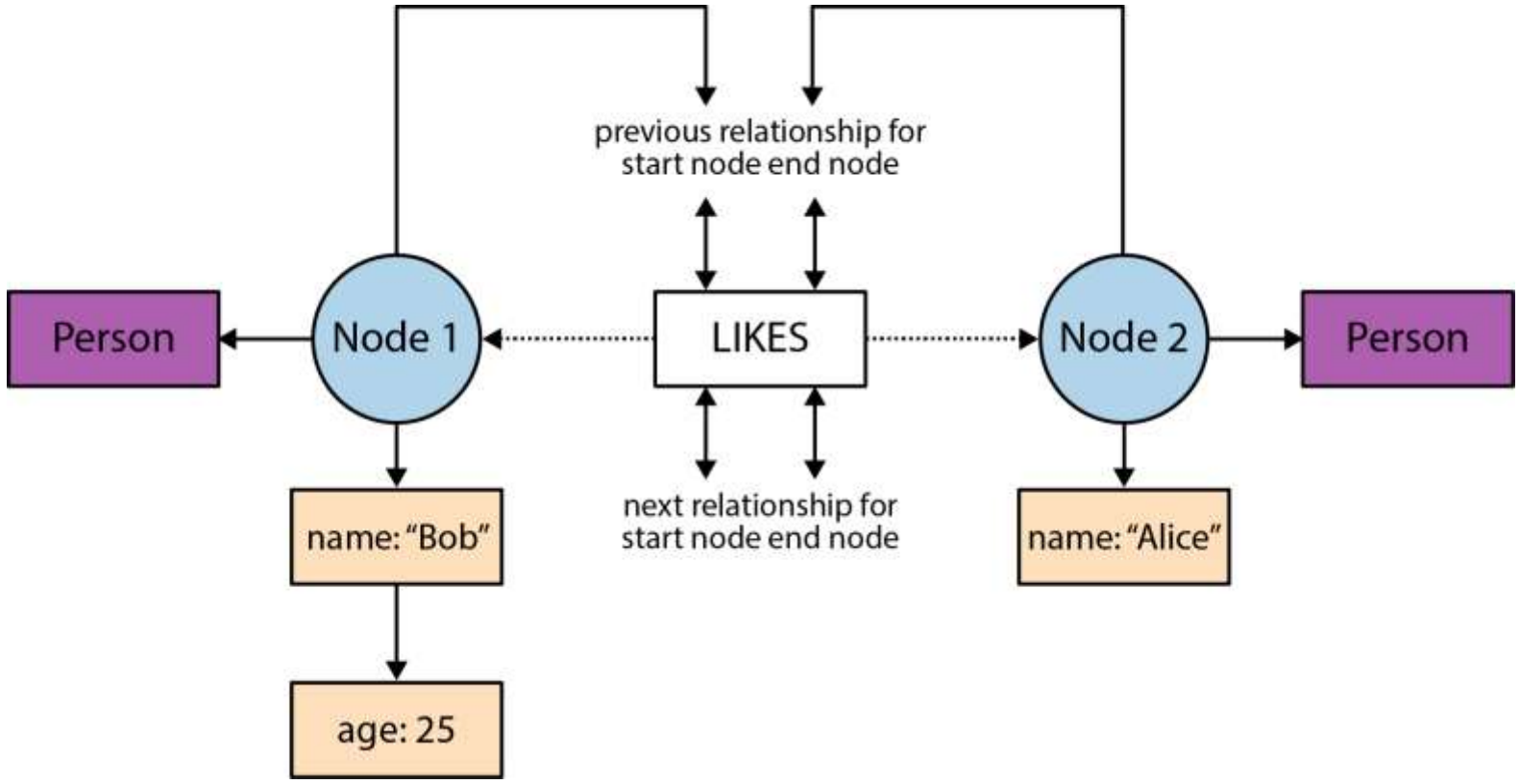
关系存储文件



- 关系边存储于独立的“关系存储文件”，每个关系边的存储空间固定，如34字节，和节点一样，这种设计便于直接通过ID编号计算获得关系边的访问地址。
- 每个节点首字节标明是否inUse，接下来四个字节存储该关系边的头节点ID，再接下来四个字节存储该关系边的尾节点ID，再接下来4个字节存储关系边类型ID，再下面存储头节点和尾节点的上一个关系边ID，以及头尾节点的下一个关系边ID。

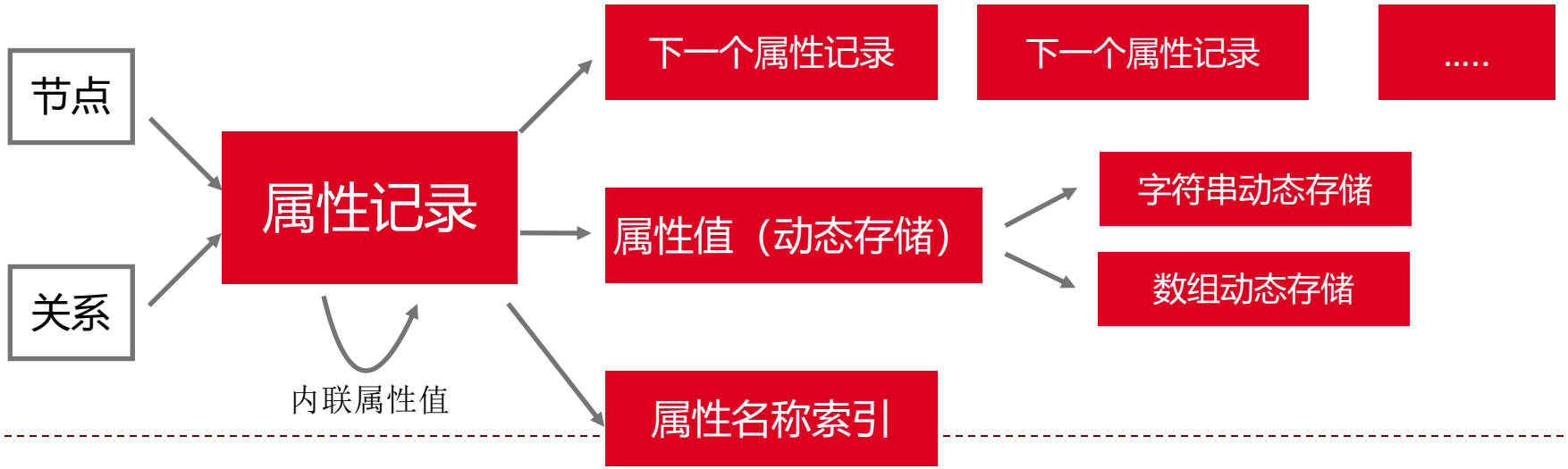
图遍历的查询的物理实现

主要优化的点：
1.节点的查找，关系的查找
2. 从节点到关系，从关系到节点
3. 从关系到关系
4. 从节点到属性，从关系到属性
5. 从关系到关系类型。



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，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）。



RDF图模型和属性图模型的比较

数据模型特性		RDF 图模型	属性图模型
结构	标准化程度	已由 W3C 制定了标准化的语法和语义[23]	尚未形成工业标准
	数学模型	3-均匀有向标签超图	有向标签属性图
	表达力	RDF 图模型强于属性图模型	属性图模型弱于 RDF 图模型
	边属性表达	通过额外方法,如“具体化”	内置支持
	概念层本体定义	RDFS[26]、OWL[27]	不支持
	串行化格式	XML[32]、JSON[33]、N-Triples[34]、Turtle[35]等	CSV
操作	查询代数	SPARQL 代数[36]	无
	查询语言	SPARQL[36]	Cypher[37]、Gremlin[38]、PGQL[39]、G-CORE[30]
约束	约束语言	RDF Shapes 约束语言(SHACL)[1]	无

from《知识图谱数据管理研究综述》王鑫等

图查询语言的比较

语法/语义/特性		SPARQL	Cypher	Gremlin	PGQL	G-CORE
图模式匹配查询	语法	CGP	CGP	CGP(无可选)[1]	CGP	CGP
	语义	子图同态、包[2]	无重复边、包[2]	子图同态、包[2]	子图同构[3]、包[2]	子图同态、包[2]
导航式查询	语法	RPQ 超集(增加反向边和属性集上的否定)	RPQ 子集(*只能作用在单边)	RPQ 超集(增加通过表达式比较属性值)	RPQ 超集(增加比较路径上的顶点和边)	RPQ 超集(增加复杂路径表达式)
	语义	任意路径、集合[4]	无重复边[5]、包[2]	任意路径[6]、包[2]	最短路径[7]、包[8]	最短路径[9]、包[2]
分析型查询		聚合函数	聚合函数	聚合函数、PageRank、PeerPressure 聚类	聚合函数	聚合函数
查询可组合性		否	是	是	否	是
数据更新语言 DML		CRUD[10]	CRUD	无	无	CR
数据定义语言 DDL		无	有	无	无	无
实现系统		Jena、RDF4J、gStore、Virtuoso 等	Neo4j、AgensGraph 等	TinkerTop 等	Oracle PGX	无

from《知识图谱数据管理研究综述》王鑫等

常见图数据库管理系统比较

类型	名称	许可证	数据模型/存储方案	查询语言	特点描述	最新版本	是否活跃
基于关系	3store	开源	RDF 图/三元组表	SPARQL	早期系统,三元组表存储方案的代表性系统	3.0.17 (2006-7-17)	否
	DLDB	研究原型	RDF 图/水平表	SPARQL	早期系统,水平表存储方案的代表性系统	已不维护	否
	Jena	开源	RDF 图/属性表	SPARQL	主流的语义 Web 工具库,RDF 数据库和 OWL 推理工具	3.10.0 (2018-12-30)	是
	SW-Store	研究原型	RDF 图/垂直划分	SPARQL	科研原型系统,垂直划分存储方案的代表性系统	已不维护	否
	IBM DB2	商业	RDF 图/DB2RDF	SPARQL/SQL	支持 RDF 的主流商业数据库	11.1 (2016-4-12)	是
	Oracle 18c	商业	RDF 图/关系存储	SPARQL/PGQL	支持 RDF 的主流商业数据库	18c (2018-2-5)	是
RDF 三元组库	RDF4J	开源	RDF 图/SAIL API	SPARQL	主流的语义 Web 工具库,RDF 数据库,提供 SAIL 接口	2.5.0 (2019-3-7)	是
	RDF-3X	开源	RDF 图/六重索引	SPARQL	科研原型系统,六重索引存储方案的代表性系统	0.3.8 (2013-11-22)	否
	gStore	开源 研究原型	RDF 图/VS*树	SPARQL	科研原型系统,原生图存储,使用了基于位串图存储技术	0.7.2 (2018-11-4)	是
	Virtuoso	商业/开源	RDF 图/多模型混合	SPARQL/SQL	语义 Web 项目常用的 RDF 数据库,基于成熟的 SQL 引擎	8.2 (2018-10-22)	是
	AllegroGraph	商业	RDF 图/三元组索引	SPARQL	对语义推理功能具有较为完善的支持	6.5.0 (2019-3-4)	是
	GraphDB	商业	RDF 图/三元组索引	SPARQL	支持语义 Web 标准的主流产品,支持 SAIL 层推理功能	8.8.1 (2019-1-30)	是
	BlazeGraph	商业	RDF 图/三元组索引	SPARQL/Gremlin	基于 RDF 三元组库的图数据库,实现了 SPARQL 和 Gremlin	2.1.4 (2016-8-30)	否
	StarDog	商业	RDF 图/三元组索引	SPARQL	对 OWL2 推理机制具有良好的支持	6.1.2 (2019-3-7)	是
原生图数据库	Neo4j	商业/开源	属性图/原生图存储	Cypher	最流行的图数据库,基于属性图模型,实现了原生优化存储	3.5.3 (2019-2-11)	是
	JanusGraph	开源	属性图 分布式存储	Gremlin	分布式图数据库,存储后端与查询引擎分离,实现了 Gremlin	0.2.2 (2018-10-9)	是
	OrientDB	商业	属性图/原生图存储	SQL/Gremlin	支持多模型的原生图数据管理系统,对数据模式的灵活支持	3.0.17 (2019-3-7)	是
	Cayley	开源	RDF 图/外部存储	Gremlin/GraphQL	轻量级开源图数据库,易于扩展对新语言和存储后端的支持	0.7.5 (2018-11-27)	是
分布式系统与框架	Sempala	开源 研究原型	RDF 图/分布式存储	SPARQL	基于 HDFS 存储,使用 Impala SQL 引擎的 RDF 三元组库	2.1 (2017-7-7)	否
	TriAD	开源 研究原型	RDF 图/ 分布式存储六重索引	SPARQL	基于 MPI 框架的异步通信协议	GitHub 源码未发布	否
	H_2RDF+	开源 研究原型	RDF 图/ 分布式存储六重索引	SPARQL	基于 HBase 构建六重索引	GitHub 源码未发布	否
	S2RDF	开源 研究原型	RDF 图/ 分布式存储垂直划分	SPARQL	基于 Spark 框架建立大量索引	1.1 (2016-4-4)	否
	Stylus	开源 研究原型	RDF 图/ 分布式存储属性表优化	SPARQL	基于分布式内存键值库的 RDF 三元组库	GitHub 源码未发布	否
	Apache Rya	开源	RDF 图/ 分布式存储三元组索引	SPARQL	基于列存储 Accumulo 的 RDF 三元组库	3.2.12 (2018-03-04)	是
	Cypher for Apache Spark	开源	属性图/ 分布式存储 DataFrame	Cypher	基于 Spark 框架的 Cypher 引擎	0.3.0 (2019-3-8)	是

from《知识图谱数据管理研究综述》王鑫等

总结：图数据存储的选择

- 图数据存储方式的选择需要综合考虑性能、动态扩展、实施成本等多方面综合因素。
- 区分原生图存储和非原生图存储：原生图存储在复杂关联查询和图计算方面有性能优势，非原生图存储兼容已有工具集通常学习和协调成本会低。
- 区分RDF图存储和属性图存储：RDF存储一般支持推理，属性图存储通常具有更好的图分析性能优势。
- 在大规模处理情况下，需要考虑与底层大数据存储引擎和上层图计算引擎集成需求。

总结：关于图模型与图数据库

- 图模型是更加接近于人脑认知和自然语言的数据模型，图数据库是处理复杂的、半结构化、多维度的、紧密关联数据的最好技术。我们鼓励在知识图谱项目中采用和实践图数据库。
- 图数据库有它弱处，假如你的应用场景不包含大量的关联查询，对于简单查询，传统关系模型和NoSQL数据库目前在性能方面更加有优势。
- RDF作为一种知识图谱表示框架的参考标准，向上对接OWL等更丰富的语义表示和推理能力，向下对接简化后的属性图模型以及图计算引擎，是最值得重视的知识图谱表示框架。

HTTP://OpenKG.CN

谢谢大家！