第八章 知识图谱

戴洪良

计算机科学与技术学院/人工智能学院

hongldai@nuaa.edu.cn

知识存储概述

- 当我们经过知识抽取得到了用于构建知识图谱的知识，就要考虑对知识的持久化存储。

- 知识图谱以图结构来对知识进行建模和表示，所以常将知识图谱中的知识作为图数据进行存储。

知识存储概述

- 实验阶段构建的小规模知识图谱多使用文件对知识进行存储

- 在面临大规模知识图谱的查询、修改、推理等需求时，需要考虑使用数据库管理系统(DBMS)对知识进行存储

- 知识存储常用的数据库管理系统有：
  - 图数据库管理系统 (Graph DBMS)
    - 基于属性图的系统
    - RDF存储系统 (RDF Stores)
  - 关系型数据库管理系统 (Relational DBMS)

图数据库管理系统

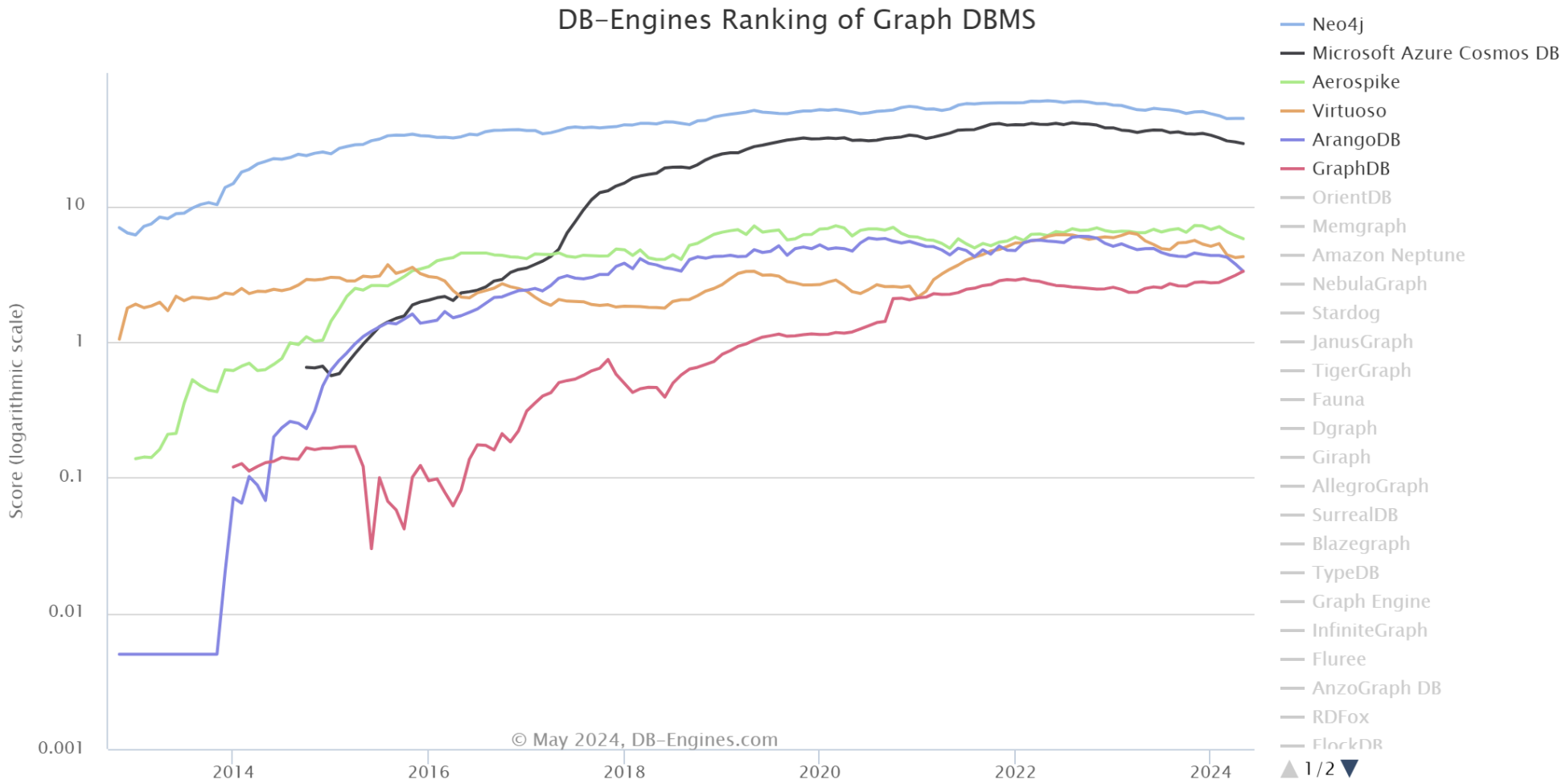
- 图结构数据、图论算法受到广泛运用，生物信息学、社交网络分析等研究领域也产生了对大规模图结构数据进行管理的需求
- 因此数据库领域发展出专门用于管理图结构数据的图数据库管理系统 (Graph Database Management System, Graph DBMS)

图数据结构模型

- 属性图（Property Graph）
  - 支持的系统有：Neo4j、Azure Cosmos DB、ArangoDB等
- RDF（Resource Description Framework）
  - 支持的系统有：Virtuoso、GraphDB、Amazon Neptune等

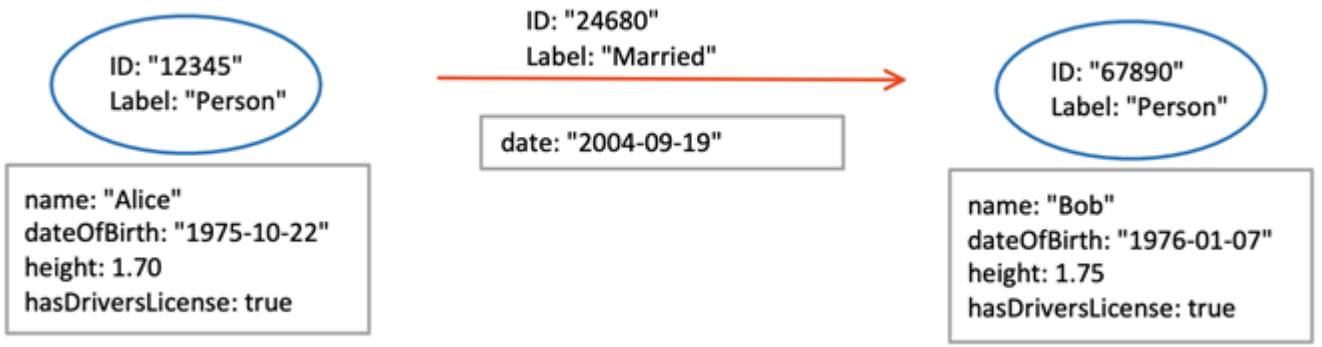
图数据库管理系统

- 受欢迎程度统计



图数据库管理系统

- 属性图(Property Graph)数据模型
  - 由节点和有向边构成；
  - 节点和边可以有标签（Labels），表示其类别；
  - 节点和边可以有属性（Properties），为键值对（key-value）形式。

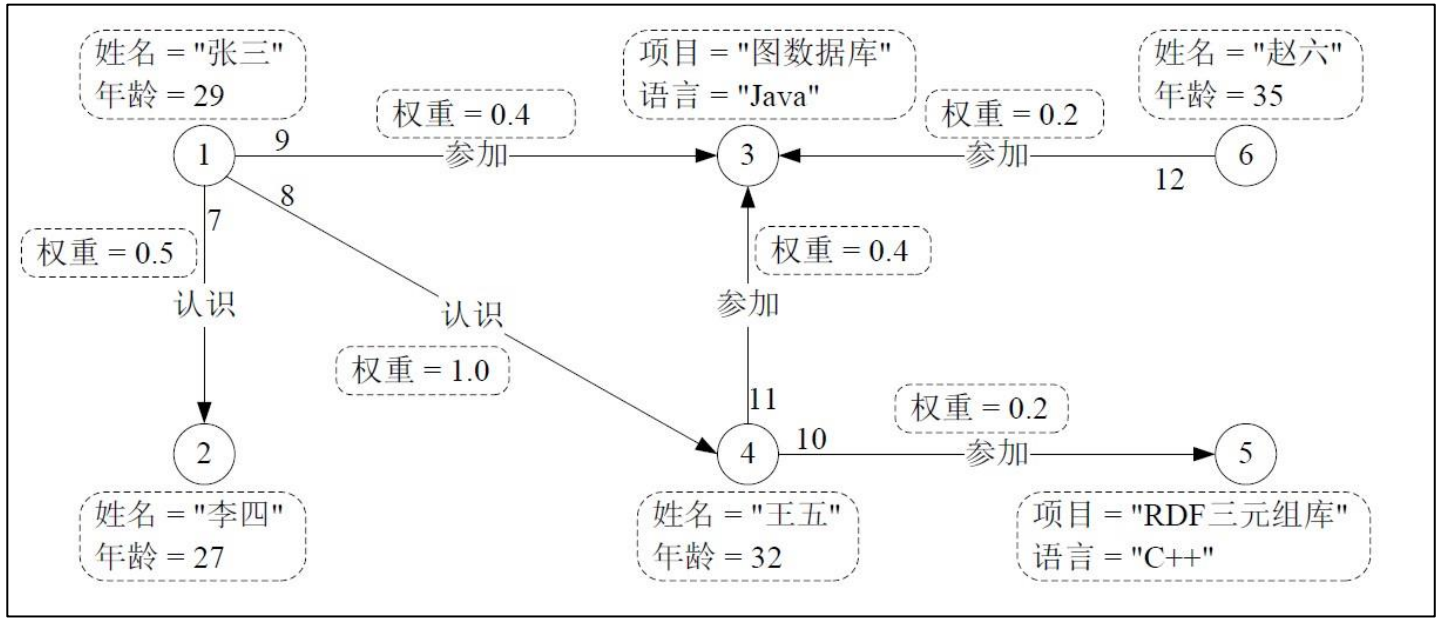


图数据库管理系统

- 基于属性图数据模型的图数据库查询语言

- Cypher
  - 声明式(declarative)语言
  - 用户只需声明“查什么”，无需关心“怎么查”
  - Neo4j图数据库专用
- Gremlin
  - 既有过程式(procedural)也有声明式(declarative)
  - 图遍历语言，如用过程式则需指明具体导航步骤
  - 业界标准， 很多图数据库都支持

图数据库查询



Gremlin: g.V(1).out('认识').where(values('年龄').is(gt(30))).out('参加').values('项目')

Cypher:
```
MATCH (n)-[:认识]->(p), (p)-[:参加]->(pr)  WHERE n.id==1, p.年龄 > 30
RETURN pr.项目
```

找出ID为1的节点认识的大于30岁的人参加过的项目

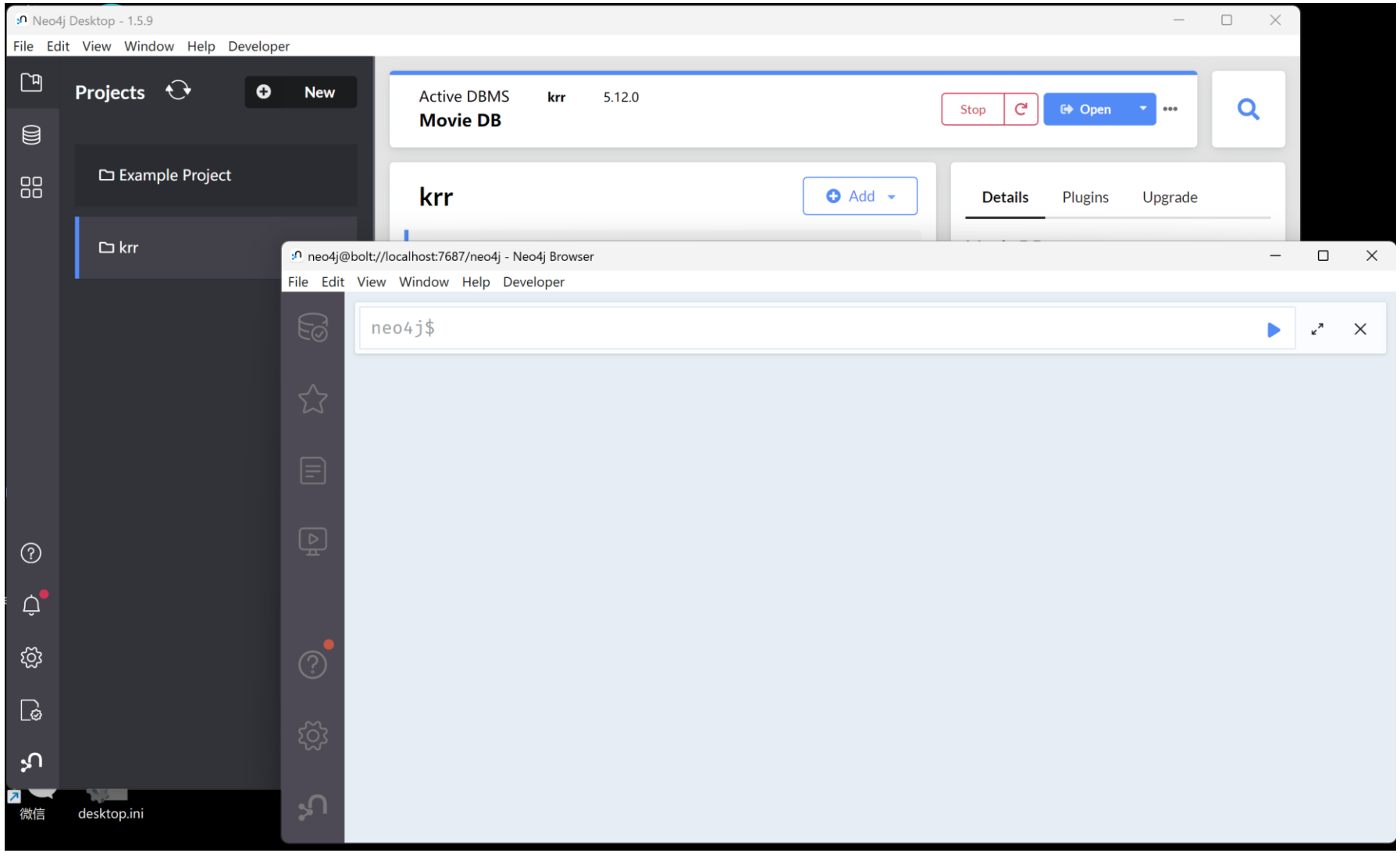
图数据库管理系统 – Neo4j

- Neo4j是原生图数据库，其使用的存储后端专门为图结构数据的存储和管理进行定制和优化

- Neo4j的数据存储形式
  - 以节点（nodes）和关系（relationships）来组织数据
    - 节点可以代表知识图谱中的实体
    - 关系可以用来代表实体间的关联，关系是有向的，两端对应开始节点和结束节点
  - 节点上可加一个或多个标签（label）表示实体的分类
  - 关系要有一个类别（type）来明确是何种类型的关系
  - 节点和关系都可以有额外描述它们的属性（properties），即键值对（key-value pairs）

Neo4j

- 桌面版界面


Neo4j Desktop - 1.5.9
File Edit View Window Help Developer
Projects
New
Example Project
krr
Active DBMS
Movie DB
krr
5.12.0
Stop
Open
krr
Add
Details
Plugins
Upgrade
neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
neo4j$
微信
desktop.ini

Neo4j - Cypher

- 添加一个节点

```
CREATE (:Movie {title:'流浪地球', released:2019})
```

- 添加一个三元组

```
CREATE (:Person {name:'郭帆', born:1980})-[:DIRECTED]->(:Movie {title:'流浪地球2', released:2023})
```

- 为已知节点添加关系

```
MATCH (we:Movie {title: '流浪地球'})
MATCH (gf:Person {name: '郭帆'})
CREATE (gf)-[:DIRECTED]->(we)
```

- 删除节点

```
MATCH (n:Movie {title: '流浪地球2'}) DETACH DELETE n
```

Neo4j - Cypher

- 添加限制

```
CREATE CONSTRAINT constraint_example_1 FOR (movie:Movie) REQUIRE movie.title IS UNIQUE
```

还可限制节点和关系的属性取值类型、属性非空等

- 添加索引

```
CREATE INDEX example_index_1 FOR (a:Actor) ON (a.name)
```

Neo4j - Cypher

- 查询5个人的名字

```
MATCH (n:Person) RETURN n.name LIMIT 5
```

- 查询90年代的电影

```
MATCH (nineties:Movie)
WHERE nineties.released > 1990 AND nineties.released < 2000
RETURN nineties.title
```

- 查询Tom Hanks演的电影

```
MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]->(tomHanksMovies)
RETURN tom,tomHanksMovies
```

Neo4j - Cypher

- ?

```
MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors)
RETURN tom, m, coActors
```

- 和Tom Hanks一起演过电影的人和相关的电影

Neo4j - Cypher

- 节点间最短路径

```
MATCH p=shortestPath(
(bacon:Person {name:"Kevin Bacon"})-[*]-(meg:Person {name:"Meg Ryan"})
)
RETURN p
```

- 通过最多两跳可到达的节点

```
MATCH (bacon:Person {name:"Kevin Bacon"})-[*1..2]-(hollywood)
RETURN DISTINCT bacon, hollywood
```

Neo4j - Python

- 用python连接数据库查询

```
from neo4j import GraphDatabase

URI = "bolt://localhost:7687"
AUTH = ("neo4j", "12345678")

with GraphDatabase.driver(URI, auth=AUTH) as driver:
    driver.verify_connectivity()

    records, summary, keys = driver.execute_query(
        "MATCH (p:Person) RETURN p.name AS name",
        database_="neo4j",
    )

    for person in records:
        print(person)
```

RDF存储系统

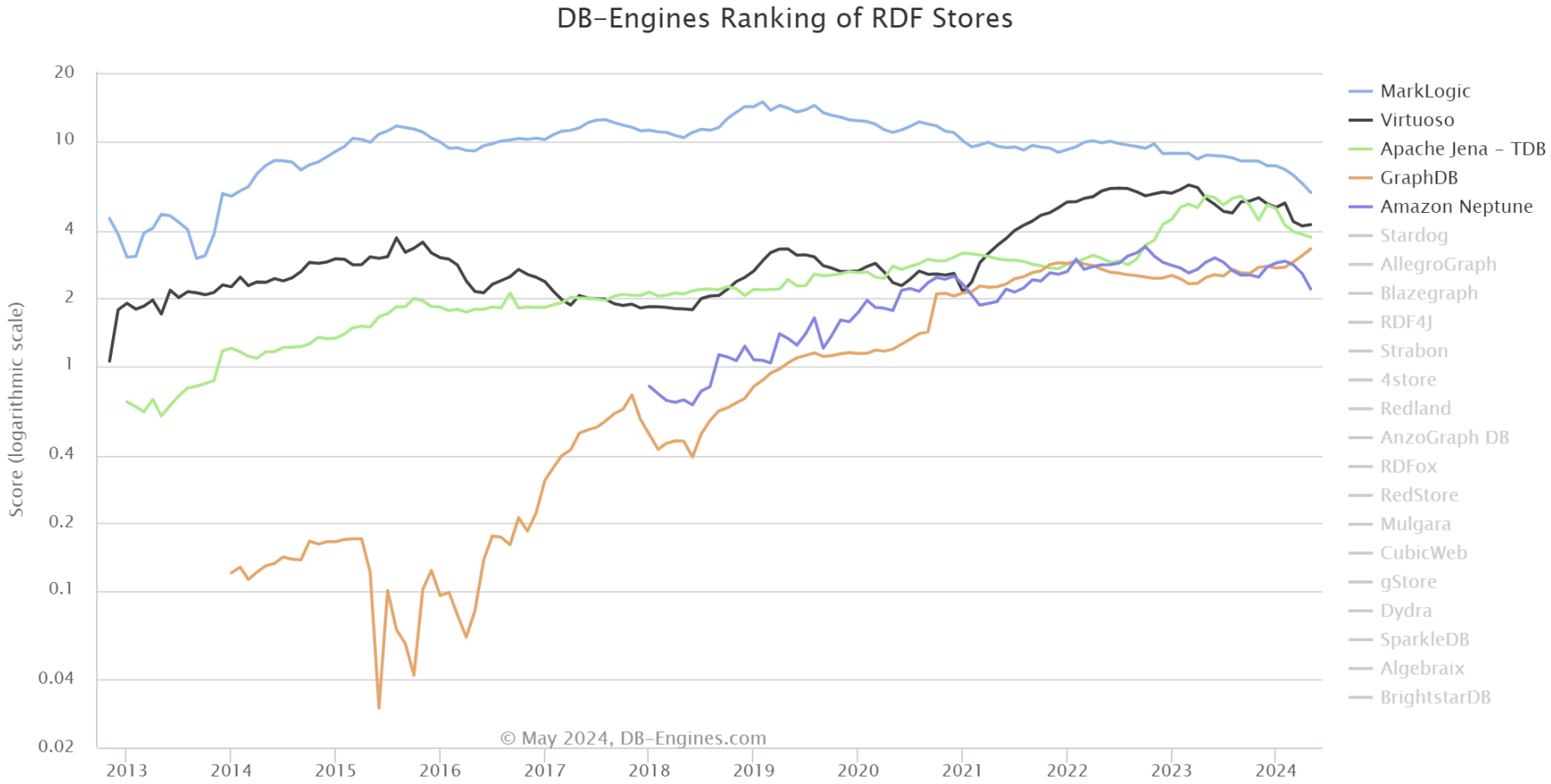
- RDF用三元组的结构来描述资源
  - 每个三元组由Subject（主语）、 Predicate（谓词） 、Object（宾语）这样的三个元素构成
  - 每一个三元组称为一个陈述（statement）
- RDF三元组的集合可看作图数据的一种表示
  - 所以RDF存储系统可以认为是一种图数据库

RDF 三元组例:

dbr:Hero_(2002_film)　dbo:director　dbr:Zhang_Yimou .

RDF存储系统

- 常见的RDF存储系统（或支持RDF存储的系统）有MarkLogic, Virtuoso, GraphDB, Amazon Neptune等



RDF存储系统

- RDF查询语言SPARQL
  - 全称为：SPARQL Protocol And RDF Query Language
  - 声明式（declarative）语言
  - 由W3C制定的RDF标准查询语言

RDF存储系统

SPARQL查询实例：

查找张艺谋导演的电影：

```
SELECT ?film, ?title
WHERE
{
  ?film dbo:director  dbr:Zhang_Yimou .
  ?film rdfs:label ?title .
  FILTER ( LANG ( ?title ) = 'zh' )
}
```

https://dbpedia.org/sparql

结果：

SPARQL | HTML5 table

film	title
http://dbpedia.org/resource/House_of_Flying_Daggers	"十面埋伏（2004年电影)"@zh
http://dbpedia.org/resource/Ju_Dou	"菊豆"@zh
http://dbpedia.org/resource/Riding_Alone_for_Thousands_of_Miles	"千里走单骑"@zh
http://dbpedia.org/resource/Under_the_Hawthorn_Tree_(film)	"山楂树之恋（电影)"@zh
http://dbpedia.org/resource/Coming_Home_(2014_film)	"归来（电影)"@zh
http://dbpedia.org/resource/Cliff_Walkers	"悬崖之上"@zh
http://dbpedia.org/resource/Codename_Cougar	"代号美洲豹"@zh
http://dbpedia.org/resource/The_Road_Home_(1999_film)	"我的父亲母亲"@zh
http://dbpedia.org/resource/The_Story_of_Qiu_Ju	"秋菊打官司"@zh
http://dbpedia.org/resource/Sniper_(2022_film)	"狙击手（电影)"@zh

基于关系型数据库的存储方案

- 在关系型数据库上设计合适的存储方案，同样能实现对图结构数据的存储。
- 关系型数据库拥有多年的发展历史，从理论到实践有着一套成熟的体系，使得一些RDF三元组库是基于关系型数据库管理系统而实现的。

基于关系型数据库的存储方案

- 三元组表
  - 将图数据用RDF三元组表示
  - 每个三元组作为表中的一行记录
  - 多跳查询时会产生自连接（Self-join）操作

如何查询生于1850年且创建过公司的人？

主语	谓词	宾语
Charles_Flint	born	1850
Charles_Flint	died	1934
Charles_Flint	founder	IBM
Larry_Page	born	1973
Larry_Page	founder	Google
…	…	…

基于关系型数据库的存储方案

- 三元组表
  - 将图数据用RDF三元组表示
  - 每个三元组作为表中的一行记录
  - 多跳查询时会产生自连接（Self-join）操作

主语	谓词	宾语
Charles_Flint	born	1850
Charles_Flint	died	1934
Charles_Flint	founder	IBM
Larry_Page	born	1973
Larry_Page	founder	Google
…	…	…

查询生于1850年且创建过公司的人：

```
SELECT t1.主语
FROM t AS t1, t AS t2
WHERE
t1.主语 = t2.主语
AND t1.谓语 = ‘born’ AND t1.宾语 = ‘1850’
   AND t2.谓语 = ‘founder’
```

基于关系型数据库的存储方案

- 水平表
  - 每行存储一个主语对应的所有宾语；
  - 只适用于谓词量较少的知识图谱；
  - 对于一个主语，可能只在极少的列上有值，导致稀疏；
  - 不易处理存储多值属性或一对多联系；

主语	born	died	founder	board	…	employees	headquarters
Charles_Flint	1850	1934	IBM		…		
Larry_Page	1973		Google	Google	…		
Android							
Google					…	54,604	Mountain_View

基于关系型数据库的存储方案

- 属性表
  - 相当于对水平表的划分；
  - 将不同类型实体存入不同的表中；
  - 适用于实体种类较少的情况；
  - 仍然不易处理多值属性及一对多联系；
  - 是一种常见的存储方案；

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	4.1	Linux	4.0

company

主语	industry	employees	headquarters
Google	Software, Internet	54,604	Mountain_View
Larry_Page	Software, Hardware, Services	433,362	Armonk

基于关系型数据库的存储方案

- 垂直划分
  - 相当于对三元组表按谓词进行划分；
  - 为每种谓词创建一张两列的表；
  - 解决了稀疏性及多值属性的问题；
  - 涉及多个谓词的查询将导致多表连接的操作；

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

developer

主语	宾语
Android	Google

version

主语	宾语
Android	4.1

kernel

主语	宾语
Android	Linux

preceded

主语	宾语
Android	4.0

industry

主语	宾语
Google	Internet
Google	Software

employees

主语	宾语
Google	57,100
IBM	377,757

headquarters

主语	宾语
Google	Mountain_View
Larry_Page	Armonk

基于关系型数据库的存储方案

- 无论使用哪种存储方案，图数据上的大多数多跳查询都会导致关系数据库中的连接（Join）操作。所以在进行图遍历时，关系数据库会产生较大开销，此时图数据库的性能优于关系型数据库。

- 但由于关系型数据库成熟的技术架构，使得其对数据的全局操作（如计数、求和）有着更为高效的性能

总结

	属性图数据库	RDF存储系统	关系型数据库
数据模型	属性图	RDF三元组	关系数据模型
查询语言	Cypher、Gremlin	SPARQL	SQL
应用场景	多为工业界场景	多为学术界场景	学术界和工业界均有应用
其他特点	图遍历效率较高;	有标准的推理引擎; 适合发布数据;	多跳查询会产生自连接操作，影响查询效率;

总结

- 在选择知识存储方式的时候，需要根据具体的数据规模及应用场景选择合适的存储方式。
- 多数实际应用场景不会涉及多跳查询（三跳以上），这时候选择传统的关系型数据库，设计合适的Schema也能满足应用需求。
- 当应用涉及多跳查询、计算最短路径、推理分析等需求时， 图数据库和RDF数据库的优势才得以体现。