3.5开源工具实践：以 gStore 为例

本节以 gStore 为例，以动手实战的模式使读者了解知识图谱数据库的各项操作，包括安装与启动、数据装载、知识图谱查询与更新等内容。

3.5.1 gStore 是什么

gStore 是由北京大学计算机科学技术研究所数据管理实验室自 2011 年开始研发的面向 RDF 知识图谱的开源图数据库系统。2013 年 12 月 gStore 的 0.1 版本完全开发完成，并于 2014 年开始在 GitHub 上进行开源。目前（2019 年 5 月），gStore 版本已经进化到 0.8 版本。在多个国际通行基准测试数据集上，当前版本 gStore 的性能已经得到了测试与验证。当前版本的 gStore 已经能够支持对包含 40 亿条边的 RDF 图构建数据库并进行秒级的查询响应。

不同于传统基于关系数据库的知识图谱数据管理方法，gStore 原生基于图数据模型，维持了原始 RDF 知识图谱的图结构；其数据模型是有标签、有向的多边图，每个顶点对应着一个主体或客体。gStore 将面向 RDF 的 SPARQL 查询，转换为面向 RDF 图的子图匹配查询，利用 gStore 所提出的基于图结构的索引来加速查询的性能。gStore 的架构如图 3-76 所示。

具体而言，gStore 将 RDF 数据解析成图格式并以邻接表的方式存储在键值数据库上。同时，gStore 将 RDF 数据上的所有点和边通过二进制编码的方式构建图索引结构。在查询处理阶段，gStore 也将 SPARQL 查询解析成查询图，并按照对 RDF 数据图的索引方式将 SPARQL 查询图进行编码。在图索引和 RDF 数据图的邻接表上，gStore 进行检索以得到每个查询变量的候选匹配。最后，gStore 将所有查询变量的候选匹配连接成最终匹配。

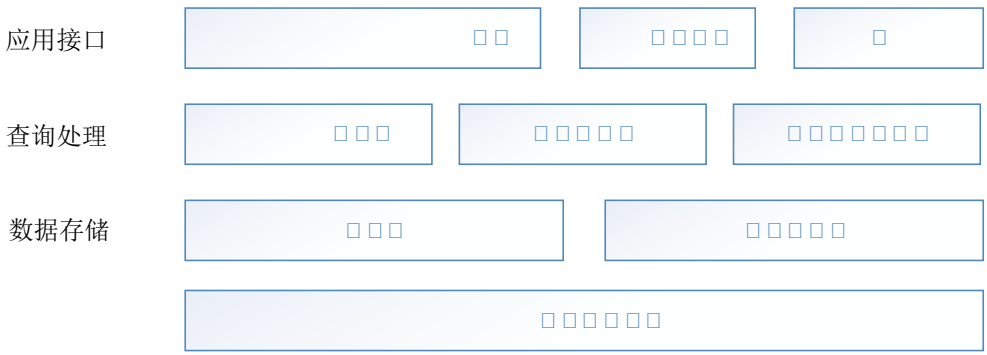


图 3-76 gStore 系统架构

gStore Workbench 是针对 gStore 设计的图数据库管理工具。在 gStore Workbench 上，用户可以创建新的数据库，载入 RDF 图数据，监控数据库状态和提交 SPARQL 查询请求。此外，gStore 还支持基于 HTTP 协议的远程图数据库访问。

3.5.2 gStore 以及 gStore Workbench 的安装

gStore 以及 gStore Workbench 都可以在 gStore 的官方网站进行下载。下面给出基于命令行和 gStore Workbench 部署 gStore 的步骤。目前，gStore 只能在 Linux 系统上通过命令行来进行部编译、部署与安装。

步骤 1。前往 gStore 官网[1]或者 GitHub 上[2]下载 gStore，解压并安装到本机的任意位置；

步骤 2。通过 make pre 和 make 命令，编译安装 gStore；

步骤 3。前往 gStore 官网上下载 gStore Workbench 安装包；

步骤 4。安装 tocmat 或者 jboss，进而部署 gStore Workbench；

步骤 5。打开浏览器，访问 http://localhost:8080/gStoreMS，出现 gStore Workbench 管理 Web 界面，如下图所示；

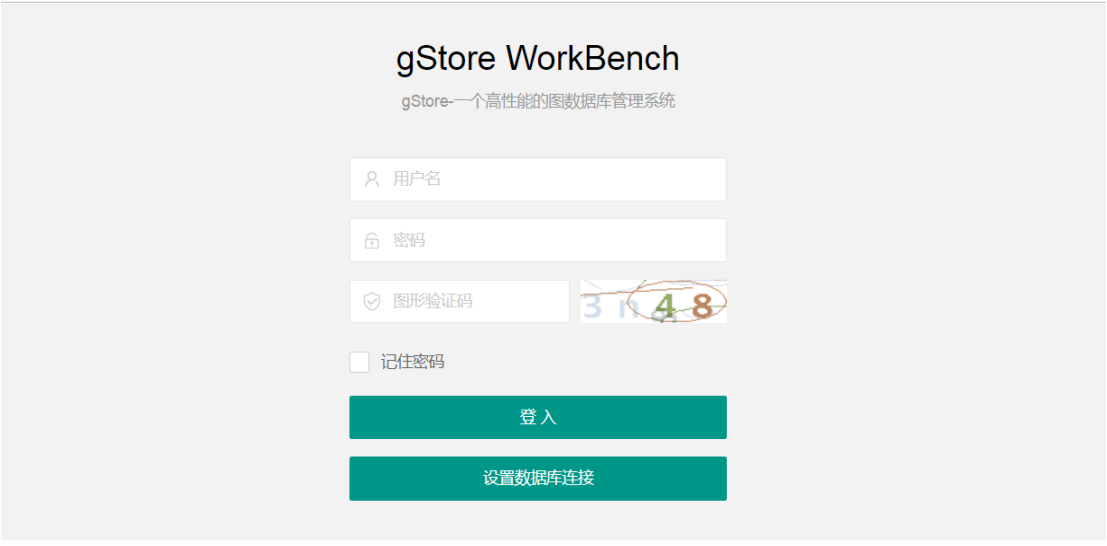


图 3-77　gStore Workbench 的登入界面

步骤 6。输入用户名、密码和验证码（默认管理员用户是 root、密码 123456），单击【登入】，进入下一个页面，如下图所示，在这个界面中就能进行数据库管理和图数据查询的功能。

[1] http://www.gstore-pku.com/
[2] https://github.com/pkumod/gStore

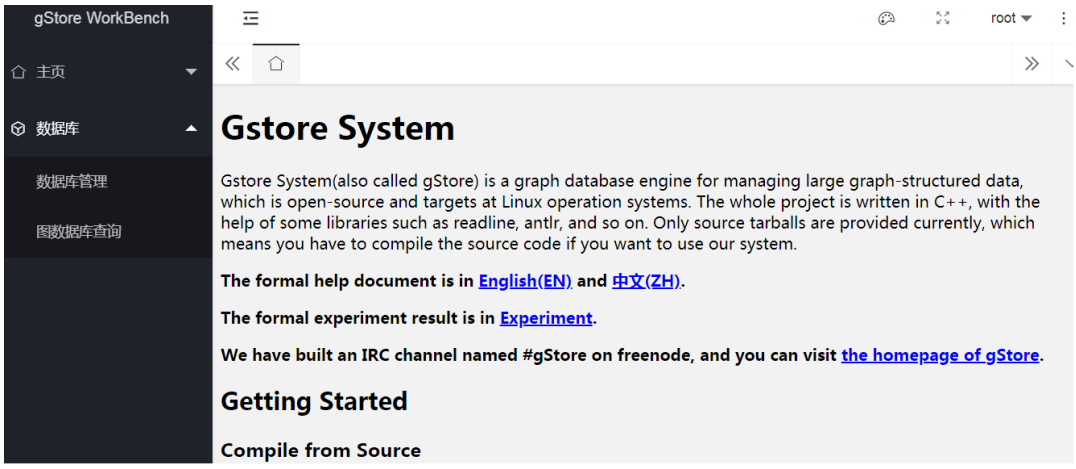


图 3-78　gStore Workbench 的数据库管理界面

3.5.3　示例知识图谱数据简介

首先本书提供一个哲学家知识图谱 Philosopher 用于实战练习。该哲学家知识图谱截取自著名知识图谱 DBpedia，可以从 gStore 官网或者 OpenKG 上进行下载，如**错误!未找到引用源。**所示：

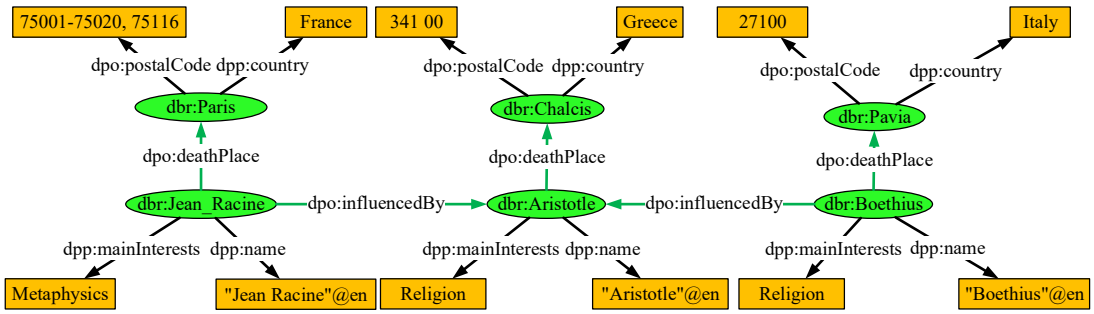


图 3-79　哲学家知识图谱

图 3-78 中包含 RDF 知识图谱中 4 种不同元素的表示：实体 RDF 资源实例（绿色椭圆），值是与实体关联的具体属性值（黄色矩形），联系是实体与另一实体之间的某种关系（绿色实线），属性连接实体与某个属性值（黑色实线）；

dpp、dpr 和 dpo 代表 DBpedia 的命名空间前缀 http://dbpedia.org/property/、http://dbpedia.org/resource/和 http://dbpedia.org/property/的缩写；

包括 6 个实体，分别是哲学家 dbr:Jean_Racine、哲学家 dbr:Aristotle、哲学家 dbr:Boethius、地点 dbr:Paris、地点 dbr:Chalcis 和地点 dbr:Pavia；

包括 2 种联系，分别是哲学家的死亡地 dpo:deathPlace、哲学家受由哪位哲学家影响 dpo:influencedBy；

包括 4 种属性：分别是哲学家的名称 dpp:name、哲学家的研究兴趣 dpp:mainInterests、

地点所在的国家 dpp:country 和地点的邮编 dpo:postalCode，这些属性均为字符串类型 xsd:string。

3.5.4 将知识图谱装载到 gStore

本小节的任务是将上述的哲学家知识图谱数据 philosopher_triples.nt 装载到 gStore Workbench 数据库。目前 gStore 支持两种方式装载数据：命令行形式和 gStore Workbench 形式。

1. 命令行形式

gStore 提供了基于命令行的数据装载命令 gbuild。使用 gStore 装载 RDF 文件 philosopher_triples.nt 的命令如图 3-80 所示，其中的 gbuild 是命令名称，testds 是数据库名称。

```
[root@iz8vb3x29nbri7buep8598z gStore]# bin/gbuild testds philosopher_triples.nt
the current settings are as below:
key : value
------------------------------------------------------------
BackupTime : 2000       # 4 am (GMT+8)
buffer_maxium : 100
db_home : .
db_suffix : .db
debug_level : simple
gstore_mode : single
operation_logs : true
thread_maxium : 1000

gbuild...
argc: 3 DB_store:testds RDF_data: philosopher_triples.nt
```

图 3-80 命令行装载数据到 gStore

2. gStore Workbench 形式

具体步骤如下：

在**错误!未找到引用源。**所示的 gStore Workbench 管理界面中单击【数据库管理】按钮，然后拖至页面最低端，点加号按钮，转到如**错误!未找到引用源。**所示的页面，单击【选择文件】按钮，浏览到 philosopher_triples.nt 文件并将其选中，单击【创建数据库】将文件上传；

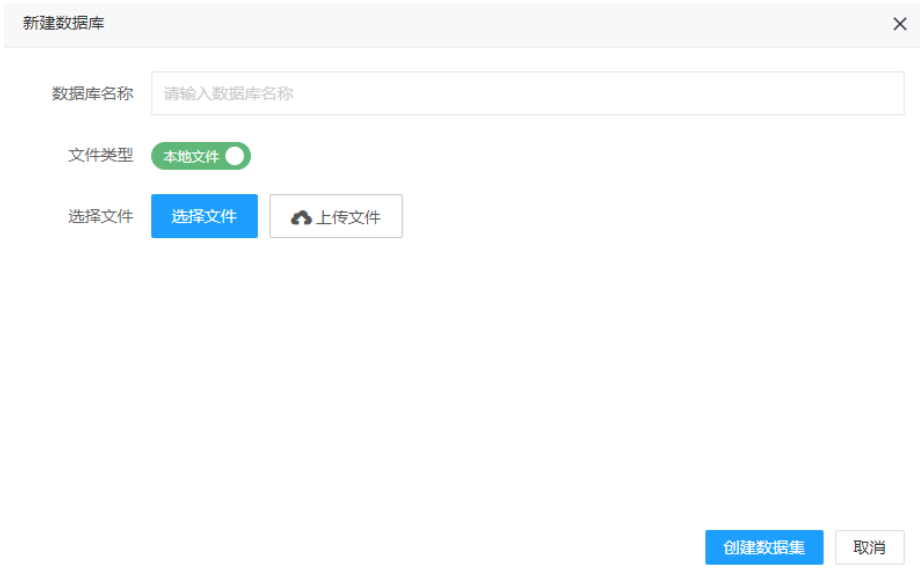


图 3-81 gStore Workbench 装载数据到 gStore

到此为止，已经完成了实战知识图谱查询与更新的所有数据准备工作。下面进入哲学家知识图谱上的实际操作。

3.5.5 查询知识图谱

本小节的任务是查询上述的 gStore 哲学家知识图谱数据库。目前 gStore 支持两种方式装载数据：命令行形式和 gStore Workbench 形式。

1. 热身准备

gStore 基于命令行形式的查询执行核心在于利用 gquery 命令，如**错误!未找到引用源。**所示。gquery 是命令名称，testds 是数据库名称，sparql01.rq 是 SPARQL 查询文件名称。

```
[root@iz8vb3x29nbri7buep8598z gStore]# bin/gquery testds sparql01.rq
the current settings are as below:
key : value
------------------------------------------------------------
BackupTime : 2000       # 4 am (GMT+8)
buffer_maxium : 100
db_home : .
db_suffix : .db
debug_level : simple
gstore_mode : single
operation_logs : true
thread_maxium : 1000

gquery...
argc: 3 DB_store:testds
                      ≀
Total time used: 90ms.
There has answer: 1
final result is :
?philosopher
<http://dbpedia.org/resource/Aristotle>
```

图 3-82　gStore 命令行的 SPARQL 查询执行命令

基于 gStore Workbench 的查询执行，首先熟悉一下用 SPARQL 查询知识图谱的界面，并执行默认的查询。具体步骤如下：

步骤1。单击**错误!未找到引用源。**左方的【图数据库查询】按钮，会进入如**错误!未找到引用源。**所示的 gStore Workbench 的 SPARQL 查询页面。

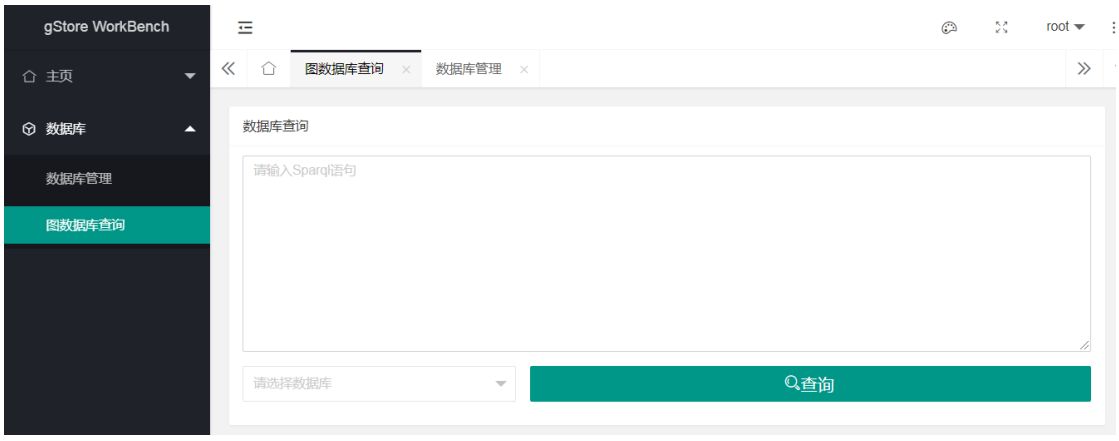


图 3-83　gStore Workbench 的 SPARQL 查询页面

步骤2。在**错误!未找到引用源。**所示的 gStore Workbench 的 SPARQL 查询界面中，“图数据库查询”部分用来输入查询，输入结束之后可以单击【查询】执行按钮提交查询；【查询】执行按钮左侧的“选择数据库”列表指定查询对应的数据集；“关系图”部分有显示查询执行的图形化结果；“返回的 JSON 数据”部分有显示 JSON 格式的查询执行结果。

步骤3。在查询数据框，单击【查询】执行按钮，在下方的 “结果关系图”部分输出

图形式的查询结果，“返回的 JSON 数据”部分输出 JSON 格式的查询结果，如**错误!未找到引用源。**所示。

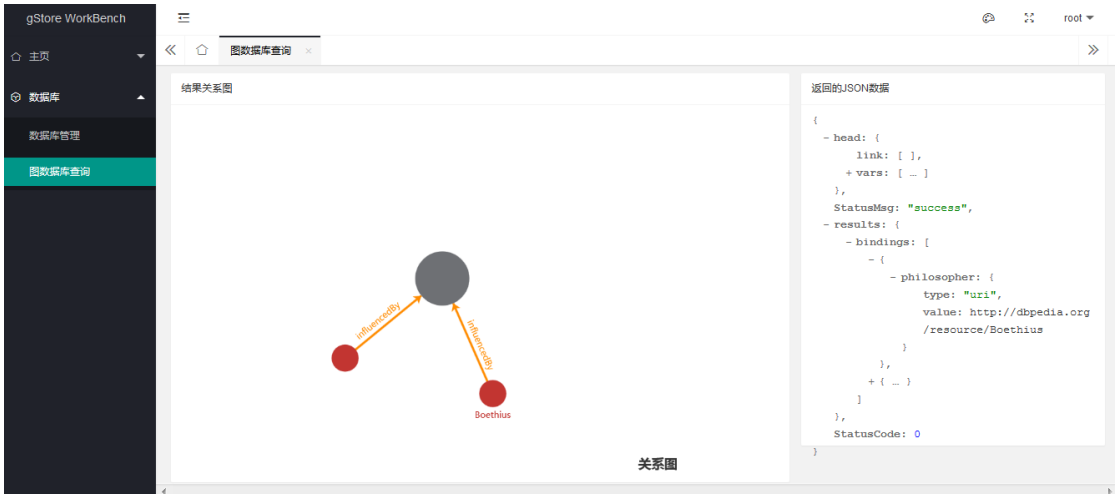
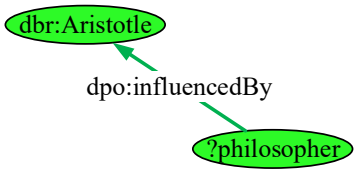


图 3-84　SPARQL 查询结果

2. 查询某一哲学家影响哪些哲学家

分析：这个查询问题要求查找哲学家，条件是某一指定哲学家影响的所有哲学家。这涉及哲学家之间的影响联系，即 dpo:influencedBy；哲学家是要求查找的、未知的实体，是变量，不妨用? philosopher 表示。某一哲学家是指定的、已知的实体，是常量，不妨假设此处某一个哲学家是 dbr:Boethius。

构造出的 SPARQL 查询图示和查询语句，如**错误!未找到引用源。**所示。显然，这是一个三元组模式查询。




```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dpo: <http://dbpedia.org/ontology/>
SELECT DISTINCT ?philosopher
WHERE {
    ?philosopher dpo:influencedBy dbr:Aristotle.
}
```

图 3-85 SPARQL 查询图示和查询语句：查询某一哲学家影响哪些哲学家

执行该查询，基于命令行和 gStore Workbench 返回的查询结果如**错误!未找到引用源。**所示。可以看到哲学家 dbr:Aristotle 总共影响了 2 个哲学家。

```
Total time used: 84ms.
There has answer: 2
final result is :
?philosopher
<http://dbpedia.org/resource/Boethius>
<http://dbpedia.org/resource/Jean_Racine>
```

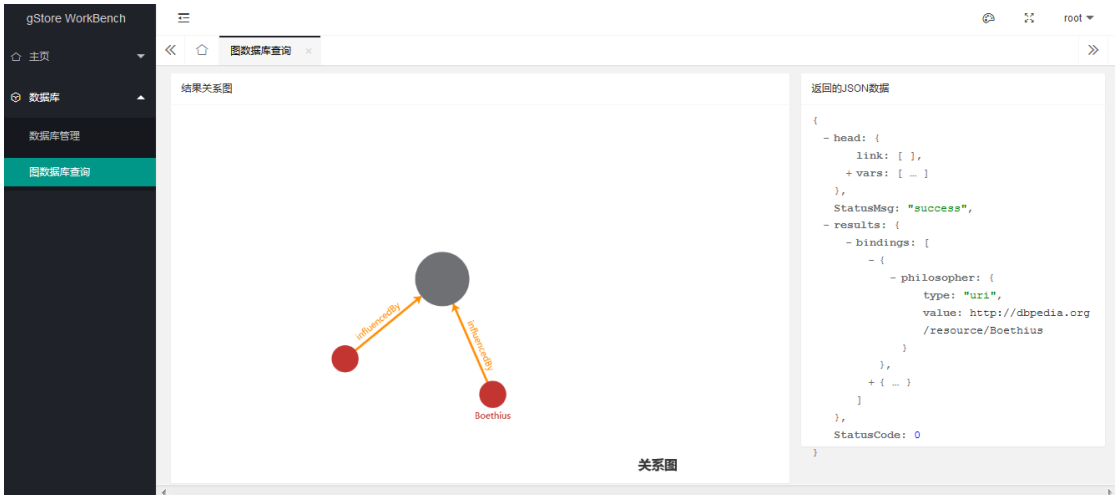


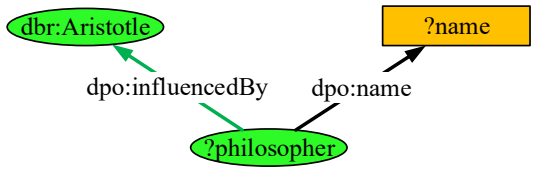
图 3-86　SPARQL 查询结果：查询哲学家 dbr:Aristotle 影响哪些哲学家

3. 还想进一步知道被影响哲学家姓名，怎么办？

如果想在上述查询返回结果的基础上进一步知道每个被影响哲学家的名称，这样的查询应该如何写呢？即查询可表达为：查询某一哲学家影响的所有哲学家和哲学家名称。

分析：只需要在上述查询基础上，为?philosopher 添加 dpp:name 属性，指向变量?name，将? philosopher 和?name 同时作为结果返回给用户。

构造出的 SPARQL 查询图示和查询语句如**错误!未找到引用源。**所示。这是一个星形的基本图模式（Basic Graph Pattern，BGP）查询。




```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dpo: <http://dbpedia.org/ontology/>
PREFIX dpp: <http://dbpedia.org/property/>
SELECT DISTINCT ?philosopher ?name
WHERE {
    ?philosopher dpo:influencedBy dbr:Aristotle.
    ?philosopher dpp:name ?name.
}
```

图 3-87　SPARQL 查询图示和查询语句：查询某一哲学家影响的所有哲学家和他们的名称

执行该查询，基于命令行和 gStore Workbench 返回的查询结果如**错误!未找到引用源。**所示。可以看到哲学家 dbr: Aristotle 总共影响的 1 位哲学家 URI 和相应的名称。

```
Total time used: 85ms.
There has answer: 2
final result is :
?philosopher    ?name
<http://dbpedia.org/resource/Boethius>  "Boethius"@en
<http://dbpedia.org/resource/Jean_Racine>       "Jean Racine"@en
```

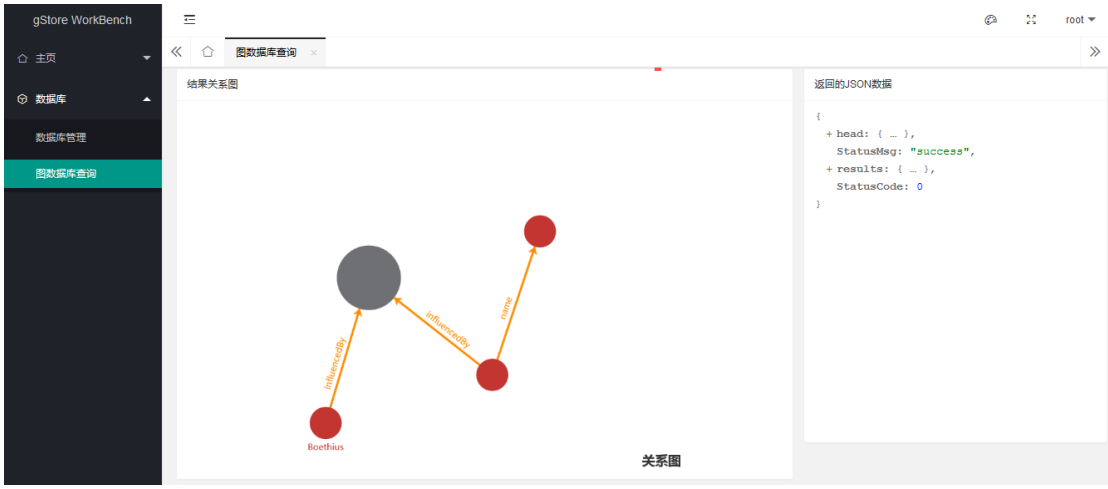
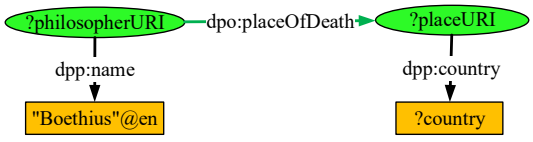


图 3-88　SPARQL 查询结果：查询哲学家 dbr:Boethius 影响哪些哲学家和他们的名称

4. 根据某一位哲学家的名字查询对应其死亡地信息

分析：设指定一位哲学家的名字"Boethius"@en，由哲学家名称出发，沿 dpp:name 属性逆向找到哲学家 URI，再由哲学家 URI 沿 dpo:deathPlace 联系找到其死亡地这个地点的 URI；要求返回的专辑信息除了哲学家 URI、死亡地 URI，还应包含死亡地所在的国家，这由死亡地这个地点 URI 的属性 dpo:country 表示。因此，要用到哲学家 URI 变量?philosopherURI、死亡地这个地点 URI 变量?placeURI 和地点所在国家变量?country。

构造出的 SPARQL 查询图示和查询语句，如**错误!未找到引用源。**所示。




```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dpo: <http://dbpedia.org/ontology/>
PREFIX dpp: <http://dbpedia.org/property/>
SELECT ?philosopherURI ?placeURI ?country
WHERE {
    ?placeURI dpp:country ?country.
    ?philosopherURI dpo:deathPlace ?placeURI.
    ?philosopherURI dpp:name "Boethius"@en.
}
```

图 3-89 SPARQL 查询图示和查询语句：查询某一个哲学家名称对应的死亡地信息

执行该查询，基于命令行和 gStore Workbench 返回的查询结果如**错误!未找到引用源。**所示。可以看到名字为"Boethius"@en 的哲学家对应的 URI、死亡地这个地点 URI 及其所在国家。

```
Total time used: 37ms.
There has answer: 1
final result is :
?philosopherURI ?placeURI       ?country
<http://dbpedia.org/resource/Boethius>  <http://dbpedia.org/resource/Pavia>     "Italy"@en
```

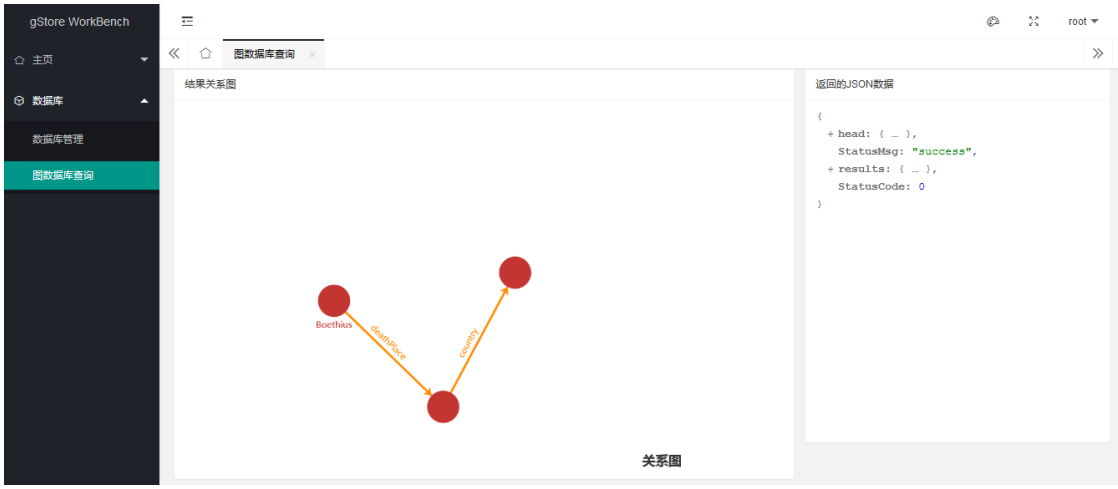


图 3-90　SPARQL 查询结果：名字为"Boethius"@en 的哲学家对应的死亡地信息

gStore 支持使用中文定义 SPARQL 变量，本查询还可以写成如**错误!未找到引用源。**所示的形式。

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dpo: <http://dbpedia.org/ontology/>
PREFIX dpp: <http://dbpedia.org/property/>
SELECT ?哲学家URI ?地点URI ?国籍
WHERE {
    ?地点URI dpp:country ?国籍.
    ?哲学家URI dpo:deathPlace ?地点URI.
    ?哲学家URI dpp:name "Boethius"@en.
}
```

图 3-91　SPARQL 查询：用中文定义变量

基于命令行和 gStore Workbench 返回的查询输出结果，如**错误!未找到引用源。**所示。

```
Total time used: 86ms.
There has answer: 1
final result is :
?哲学家URI      ?地点URI        ?国籍
<http://dbpedia.org/resource/Boethius>  <http://dbpedia.org/resource/Pavia>     "Italy"@en
```

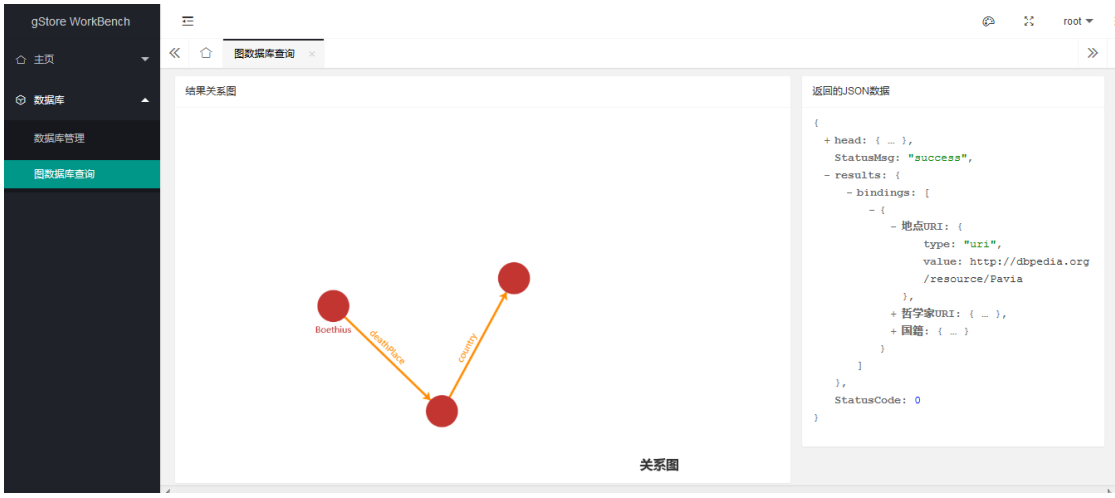
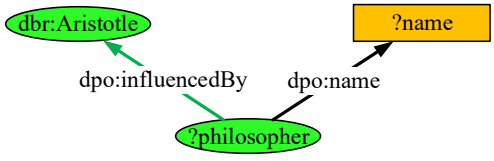


图 3-92　SPARQL 查询结果：用中文定义变量

5. 基于 LIMIT 查询受某位哲学家影响的所有哲学家

分析：设指定哲学家名称为"Aristotle"@en，由哲学家名称出发，沿 dpp:name 属性逆向找到哲学家 URI，再由哲学家 URI 沿 dpo: influencedBy 联系逆向找到被该哲学家影响的哲学家 URI。如果只需要查找该专辑里面的前 2 首歌曲，应该如何修改该查询语句呢？需要使用 LIMIT 关键字对查询结果条数进行限制。构造出的 SPARQL 查询图示和查询语句，如**错误!未找到引用源。**所示。




```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dpo: <http://dbpedia.org/ontology/>
PREFIX dpp: <http://dbpedia.org/property/>
SELECT ?philosopher
WHERE {
    ?philosopher dpo:influencedBy dbr:Aristotle.
    ?philosopher dpp:name ?name.
}
LIMIT 1
```

图 3-93　SPARQL 查询图示和查询语句：查询受某位哲学家影响的所有哲学家并使用 LIMIT 限制查询结果条数

如果想知道被影响哲学家的确切数目，可使用聚合函数 COUNT 对哲学家进行计数，如**错误!未找到引用源。**所示。

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dpo: <http://dbpedia.org/ontology/>
PREFIX dpp: <http://dbpedia.org/property/>
SELECT (COUNT(?philosopher) AS ?num)
WHERE {
    ?philosopher dpo:influencedBy dbr:Aristotle.
    ?philosopher dpp:name ?name.
}
```

图 3-94　SPARQL 查询语句：使用聚合函数 COUNT 进行计数

6. 基于 ORDER BY 查询并排序受某位哲学家影响的所有哲学家

分析：该查询也与查询 5 的模式相同。但是另一个需求来了！用户提出要给结果按姓名排序。使用 ORDER BY 关键字完成该任务。只需在上面查询最后加上 ORDER BY ?name 子句，即可将结果按照标签的升序进行排列，如**错误!未找到引用源。**所示。如果想按照标签降序进行逆向排序，可换为使用 ORDER BY DESC(?tag_name)子句。

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dpo: <http://dbpedia.org/ontology/>
PREFIX dpp: <http://dbpedia.org/property/>
SELECT ?name
WHERE {
    ?philosopher dpo:influencedBy dbr:Aristotle.
    ?philosopher dpp:name ?name.
}
ORDER BY ?name
```

图 3-95　SPARQL 查询：使用 ORDER BY 关键字进行排序

7. 查询某几个国家地点的数目

分析：首先，求地点数目是一个统计查询，需要用到聚合函数 COUNT。不妨设求出国家"Italy"@en 和"France"@en 的地点数目。一种方法是分别求出每个国家的所有地点，然后对这两个集合的地点求并集操作；另一种方法是采用 FILTER 关键字将两个地点设定为国家的过滤条件。

构造出相应的 SPARQL 查询语句，如**错误!未找到引用源。**所示。

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dpo: <http://dbpedia.org/ontology/>
PREFIX dpp: <http://dbpedia.org/property/>
SELECT ?placeURI
WHERE {
    { ?placeURI dpp:country     "France"@en. }
    UNION
    { ?placeURI dpp:country     "Italy"@en. }
}
```

图 3-96　SPARQL 查询语句：查询某几类歌曲标签中的歌曲数目

使用 FILTER 关键字的等价 SPARQL 查询，如**错误!未找到引用源。**所示。

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dpo: <http://dbpedia.org/ontology/>
PREFIX dpp: <http://dbpedia.org/property/>
SELECT ?placeURI ?country
WHERE {
    ?placeURI dpp:country       ?country
    FILTER (?country = "France"@en || ?country = "Italy"@en)
}
```

图 3-91　SPARQL 查询：使用 FILTER 关键字进行过滤

8. 询问是否存在含有某字符串的哲学家名

分析：该查询不同于之前的所有查询，这是一个 Yes/No 问题。在 SPARQL 中应该使用 ASK 关键字而不是 SELECT 引导该查询。而是否含有某字符串的判断可以使用正则表

达式匹配函数 regex 完成。

构造相应的 SPARQL 查询语句，如**错误!未找到引用源。**所示。

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dpo: <http://dbpedia.org/ontology/>
PREFIX dpp: <http://dbpedia.org/property/>
ASK {
    ?philosopher dpp:name       ?name .
    FILTER regex(?name, "Racine")
}
```

图 3-98　SPARQL 查询语句：询问是否存在含有某字符串的哲学家名

ASK 查询的结果只有 True 或者 False，例如上述查询基于命令行返回的结果，如**错误!未找到引用源。**所示。gStore Workbench 目前支持 ASK 查询，但是无返回结果。

```
Total time used: 84ms.
There has answer: 1
final result is :
?_askResult
true
```

图 3-99　SPARQL 查询：ASK 查询的返回结果

3.5.6更新知识图谱

经过前一小节若干查询实例的练习，想必读者已经基本掌握了使用 SPARQL 进行知识图谱查询的一些要点。本小节一起学习如何使用 SPARQL 完成更新已有的知识图谱。注意：gStore Workbench 也支持更新，但是无返回结果。

1. 给地点 URI 新增属性歌手名字

分析：类似于哲学家实体，地点实体也是可以有名称 dpp:name 的。下面通过 INSERT 关键字向哲学家知识图谱中插入 3 个地点实体的名字。gStore 中也是通过 gquery 命令来实现这项功能。

给 URI 为 dbr:Paris、dbr:Pavia 和 dbr:Chalcis 的地点分别添加名字"Paris"@en、"Pavia"@en 和"Chalcis"@en，构造 SPARQL 更新语句，如**错误!未找到引用源。**所示。

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dpo: <http://dbpedia.org/ontology/>
PREFIX dpp: <http://dbpedia.org/property/>
INSERT DATA {
    dbr:Paris dpp:name "Paris"@en.
    dbr:Pavia dpp:name "Pavia"@en.
    dbr:Chalcis dpp:name "Chalcis"@en.
}
```

图 3-100　SPARQL 更新语句：给地点实体 URI 新增属性名称

执行该更新操作，基于命令行返回的执行结果如下图所示。

```
Total time used: 82ms.
update num: 3
```

图 3-101　SPARQL 更新结果：给地点实体 URI 新增属性名称

这时，可以执行一个 SPARQL 查询，查看刚刚更新的数据是否产生了作用，该查询的编写和执行留给读者作为练习。

2. 删除某个地点的邮编属性

分析：首先使用 WHERE 子句定位要删除的三元组，然后使用 DELETE 子句完成删除操作。

删除地点 dbr:Paris 的邮编属性，构造 SPARQL 更新语句，如**错误!未找到引用源。**所示。

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dpo: <http://dbpedia.org/ontology/>
PREFIX dpp: <http://dbpedia.org/property/>
DELETE {
    dbr:Paris dpo:postalCode ?x.
}
WHERE {
    dbr:Paris dpo:postalCode ?x.
}
```

图 3-102　SPARQL 更新语句：删除增加的属性地点邮编

执行该更新操作，基于命令行返回的执行结果如下图所示。

```
Total time used: 83ms.
update num: 1
```

图 3-103　SPARQL 更新结果：给地点实体 URI 新增属性名称

也可以执行一个查询，以验证符合条件的三元组 dbr:Paris dpo:postalCode ?x 已经被删除，该查询的编写和执行也留给读者作为练习。

参考文献

[1] Graham Klyne，Jeremy J Carroll， Brian McBride. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation.（2014-2-25）. https://www.w3.org/TR/rdf11-concepts/.

[2] Tim Berners-Lee. Reification of RDF and N3 - Design Issues.（2004-12-17）.https://www.w3.org/DesignIssues/Reify.html.

[3] Tinkerpop/Blueprints. Property Graph Model. （2016-7-13）.https://github.com/tinkerpop/blueprints/wiki/Property-Graph-Model.

[4] Steve Harris，Andy Seaborne. SPARQL 1.1 Query Language.W3C Recommendation 21 March 2013. https://www.w3.org/TR/sparql11-query/.

[5] Wikipedia. Subgraph isomorphism problem.（2018-10-13）.https://en.wikipedia.org/wiki/Subgraph_isomorphism_problem.

[6] Paula Gearon，Alexandre Passant，Axel Polleres. SPARQL 1.1 Update. W3C Recommendation 21 March 2013. https://www.w3.org/TR/sparql11-update/.

[7] Gregory Todd Williams. SPARQL 1.1 Service Description. W3C Recommendation 21 March 2013. https://www.w3.org/TR/sparql11-service-description/.

[8] Eric Prud'hommeaux，Carlos Buil-Aranda. SPARQL 1.1 Federated Query. W3C Recommendation 21 March 2013. https://www.w3.org/TR/sparql11-federated-query/.

[9] Andy Seaborne. SPARQL 1.1 Query Results JSON Format. W3C Recommendation 21 March 2013. http://www.w3.org/TR/sparql11-results-json/.

[10] Birte Glimm，Chimezie Ogbuji. SPARQL 1.1 Entailment Regimes. W3C Recommendation 21 March 2013. http://www.w3.org/TR/sparql11-entailment/.

[11] Lee Feigenbaum，Gregory Todd Williams，Kendall Grant Clark，et al. SPARQL 1.1 Protocol. W3C Recommendation 21 March 2013. http://www.w3.org/TR/sparql11-protocol/.

[12] WikiBooks. SPARQL. https://en.wikibooks.org/wiki/SPARQL.

[13] Wikidata. SPARQL tutorial. https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial.

[14] Apache Jena. SPARQL Tutorial. https://jena.apache.org/tutorials/sparql.html.

[15] Neo4j. Cypher Query Language Developer Guides & Tutorials. https://neo4j.com/developer/cypher/.

[16] The openCypher Project. https://www.opencypher.org.

[17] Michael Rudolf，Marcus Paradies，Christof Bornhövd，et al. The Graph Story of the SAP HANA Database.BTW，2013，13：403–420.

[18] RedisGraph. https://oss.redislabs.com/redisgraph/.

[19] AgensGraph - The Performance-Driven Graph Database. （2017）. http://www.agensgraph.com/.

[20] Memgraph. https://memgraph.com/.

[21] Apache TinkerPop. TinkerPop3 Documentation v.3.3.3.（2018-8）.http://tinkerpop.apache.org/docs/3.3.3/reference/.

[22] Apache TinkerPop. The Gremlin Graph Traversal Machine and Language. https://tinkerpop.apache.org/gremlin.html.

[23] The DBpedia Project. https://wiki.dbpedia.org/.

[24] Harris S，Gibbins N. 3store: Efficient bulk RDF storage. 1st International Workshop on Practical and Scalable Semantic Systems ，2003.

[25] PAN Zh X，Heflin J. DLDB:Extending Relational Databases to Support Semantic Web Queries. Proceedings of the First International Workshop on Practical and Scalable Semantic Systems，2003.

[26] Wilkinson K. Jena Property Table Implementation. The Second International Workshop on Scalable Semantic Web Knowledge Base，2016.

[27] Abadi D J，Marcus A，Madden S R，et al. Scalable Semantic Web Data Management Using Vertical Partitioning//Proceedings of the 33rd international conference on Very large data bases. VLDB Endowment，2007：411-422.

[28] Abadi D J, Marcus A, Madden S R. SW-Store: a vertically partitioned DBMS for Semantic Web data management. Vldb Journal, 2009, 18(2):385-406.

[29] Neumann T，Weikum G. RDF-3X: a RISC-style engine for RDF. Proceedings of the VLDB Endowment，2008，1（1）：647-659.

[30] Weiss C，Karras P，Bernstein A. Hexastore: Sextuple Indexing for Semantic Web Data Management. Proceedings of the VLDB Endowment，2008，1（1）：1008-1019.

[31] Bornea M A，Dolby J，Kementsietsidis A，et al. Building an Efficient RDF Store Over a Relational Database//Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM，2013：121-132.
[32] Wikipedia. Graph coloring. https://en.wikipedia.org/wiki/Graph_coloring.
[33] https://en.wikipedia.org/wiki/Greedy_coloring.
[34] Zou L，Özsu M T，Chen L，et al. gStore：A Graph-Based Sparql Query Engine. The VLDB journal，2014，23（4）：565-590.
[35] W3C Wiki. LargeTripleStores. https://www.w3.org/wiki/LargeTripleStores.
[36] Ian Robinson，Jim Webber，Emil Eifrem. Graph Databases. . O'Reilly Media，2015.
[37] Giugno R，Shasha D. Graphgrep：A Fast and Universal Method for Querying Graphs//Pattern Recognition. Proceedingsof the 16th International Conference on IEEE，2002，2：112-115.
[38] Yan X，Yu P S，Han J. Graph Indexing: A Frequent Structure-Based Approach//Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data. ACM，2004：335-346.
[39] Apache Jena. https://jena.apache.org/.
[40] Apache Jena Fuseki. https://jena.apache.org/documentation/fuseki2/.
[41] Chimezie Ogbuji. SPARQL 1.1 Graph Store HTTP Protocol. W3C Recommendation 21 March 2013. https://www.w3.org/TR/sparql11-http-rdf-update/.