

3.1 开源工具实践：以 Apache Jena 为例

本节以 Apache Jena 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。

3.1.1 Jena 是什么

Apache Jena 的前身是惠普实验室（HP Labs）2000 年开发的 Jena 工具包^{错误:未找到引用源。}。2010 年 11 月，惠普开发团队将 Jena 移交到 Apache 软件基金会旗下继续进行开发。2012 年 4 月，Jena 升级为 Apache 顶级项目。Jena 从发布起就一直是语义 Web 领域最为流行的开源 Java 框架和 RDF 数据库之一，并始终遵循 W3C 标准，其提供的 API 功能包括：RDF 数据管理、RDFS 和 OWL 本体管理、SPARQL 查询处理。Jena 具备一套基于规则的推理引擎用以执行 RDFS 和 OWL 本体上的推理任务。从知识存储的角度来看，Jena 具备一套原生的存储引擎，可以高效地对 RDF 三元组数据进行基于磁盘或内存的存储管理。

Apache Jena 框架的架构图如图 3-1 所示，图中给出了框架各个 API 之间的交互和调用关系。自底向上看，Jena 的存储 API 为上层提供基本三元组存储和本体存储功能，支持的底层存储类型包括基于内存的存储、基于关系数据库的 SDB 存储、基于原生三元组的 TDB 存储和用户定制的存储。推理 API 为上层提供本体推理服务，如果为了追求查询处理效率且不使用推理功能，可以选择不设置推理机，也可以使用 Jena 内置的基于规则的推理机进行 RDFS 和 OWL 本体上的推理任务，或者选择通过接口调用第三方外部推理机。Jena 对外界应用程序的 API 包括实现基本三元组管理功能的 RDF API、实现 RDFS 和 OWL 本体推理功能的本体 API 和实现查询处理功能的 SPARQL API。Java 应用程序代码可以通过导入类库的形式直接调用这些 API。此外，Jena 还提供了支持各种 RDF 三元组格式的解析器和编写器，支持的三元组格式包括 RDF/XML、Turtle、N-Triple 和 RDFa。实质上，Jena 是一个 Java 框架类库。在一般情况下，上述功能需要在 Java 程序中进行调用。不过，Jena 为了用户使用方便，还提供了一个名为 Fuseki 的独立的 RDF 数据库 Web 应用程序。本节正是要使用 Fuseki 作为实战知识图谱数据库的入门工具。

Fuseki 是基于 Jena 的 SPARQL 服务器，其可以作为一个独立的服务由命令行启动，也可以作为一个操作系统服务或 Java Web 应用程序^{错误:未找到引用源。}。Fuseki 底层存储基于 TDB，具有 SPARQL 查询处理的 Web 用户界面，同时提供服务器监控和管理功能界面。Fuseki 支持最新的 SPARQL 1.1 版本^{错误:未找到引用源。}，同时支持 SPARQL 图存储 HTTP 协议^{错误:未找到引用源。}。

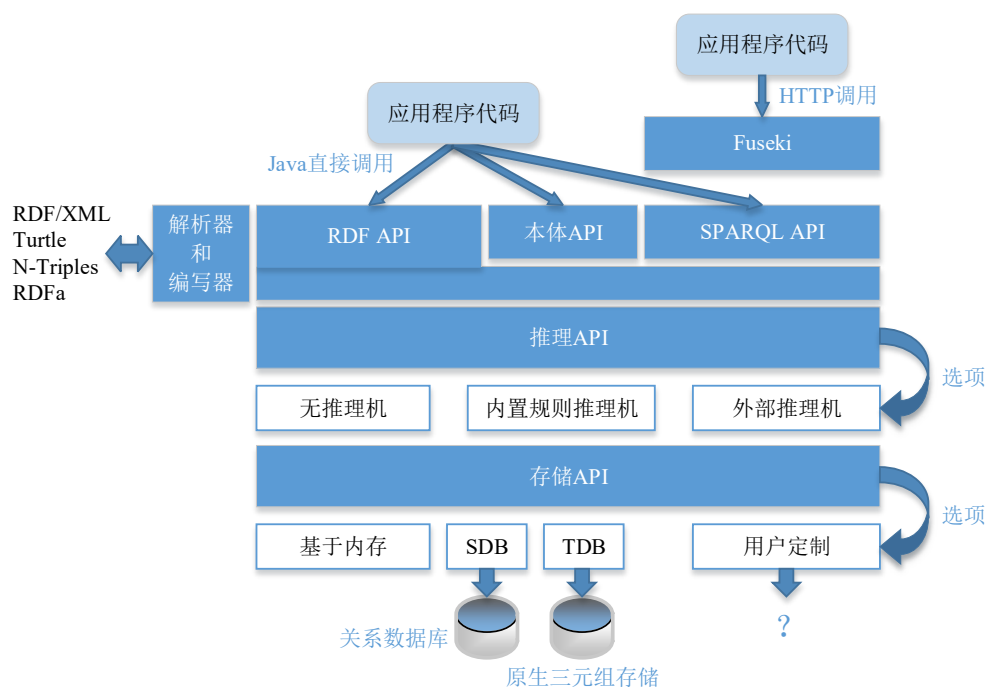


图 3-1 Apache Jena 框架的架构图

3.1.2 Jena Fuseki 的安装

下面是 Jena Fuseki 的安装步骤：

步骤 1。前往 Jena 官网下载最新版本的 Fuseki 压缩包文件 `apache-jena-fuseki-3.6.0.zip`（本书写作时的最新版本为 3.6.0）；

步骤 2。将上述压缩包解压到本机的任意目录，该目录即为 Fuseki 的安装目录；例如，解压到目录 `C:\tools\apache-jena-fuseki-3.6.0\`；如图 3-2 所示。

步骤 3。Fuseki 的运行需要 Java 环境的支持，版本要求是 JDK 8 以上，如机器中没有安装 JDK 或版本不满足要求，请安装。

此电脑 > Windows (C:) > tools > apache-jena-fuseki-3.6.0			
名称	修改日期	类型	大小
bin	2017/12/13 21:39	文件夹	
webapp	2017/12/13 21:13	文件夹	
fuseki	2017/12/13 21:13	文件	13 KB
fuseki.war	2017/12/13 21:35	WAR 文件	23,353 KB
fuseki-server	2017/12/13 21:13	文件	3 KB
fuseki-server.bat	2017/12/13 21:13	Windows 批处理文件	2 KB
fuseki-server.jar	2017/12/13 21:36	Executable Jar File	26,071 KB
LICENSE	2017/12/13 21:13	文件	30 KB
NOTICE	2017/12/13 21:13	文件	10 KB
README	2017/12/13 21:13	文件	3 KB

图 3-2 Jena Fuseki 的安装目录

3.1.3 启动 Fuseki

启动 Jena Fuseki 的步骤如下：

步骤1。运行“命令提示符”，进入命令行；

步骤2。进入目录 Fuseki 安装目录，执行命令：

```
cd C:\tools\apache-jena-fuseki-3.6.0
```

步骤3。在该目录下新建文件夹 **store**，用于保存 Jena TDB 数据库文件，执行命令：
`mkdir store`

步骤4。启动 Fuseki，执行命令：
`fuseki-server --loc=store --update /testds`
其中，命令行选项 `--loc` 指定数据库目录为上一步新建的文件夹 **store**；命令行选项 `--update` 表示支持 SPARQL Update 数据更新操作；参数 `/testds` 是指定默认数据集的名称；命令行输入和输出如图 3-3 所示：

● Fuseki 具有一个内置的 Web 服务器，注意，该命令行窗口作为后台服务器保留，不要关闭；使用浏览器打开 Fuseki Web 用户界面，地址为 `http://localhost:3030/`，如图 3-4 所示。可以看到：

右上角的服务器状态（**Server status**）指示灯为绿色，表示当前服务器状态正常；使用的 Fuseki 版本（**Version**）为 3.6.0，服务器在线时间（**Uptime**）为 3 分 49 秒（3m 49s）；服务器上的数据集（**Datasets on this server**）就有 `/testds`，这个数据集就是启动 Fuseki 服务器时指定的默认数据集。

```
命令提示符 - fuseki-server --loc=store --update /testds
C:\Users\WX>cd C:\tools\apache-jena-fuseki-3.6.0
C:\tools\apache-jena-fuseki-3.6.0>mkdir store
C:\tools\apache-jena-fuseki-3.6.0>fuseki-server --loc=store --update /testds
[2018-01-27 17:20:06] Server INFO Running in read-only mode for /testds
[2018-01-27 17:20:06] Server INFO Apache Jena Fuseki 3.6.0
[2018-01-27 17:20:06] Config INFO FUSEKI_HOME=C:\tools\apache-jena-fuseki-3.6.0\
[2018-01-27 17:20:06] Config INFO FUSEKI_BASE=C:\tools\apache-jena-fuseki-3.6.0\run
[2018-01-27 17:20:06] Config INFO Shiro file: file://C:\tools\apache-jena-fuseki-3.6.0\run\shiro.ini
[2018-01-27 17:20:07] Config INFO Template file: templates/config-tdb-dir
[2018-01-27 17:20:07] Config INFO TDB dataset: directory=store
[2018-01-27 17:20:07] Config INFO Register: /testds
[2018-01-27 17:20:07] Server INFO Started 2018/01/27 17:20:07 CST on port 3030
```

图 3-3 命令行输入和输出

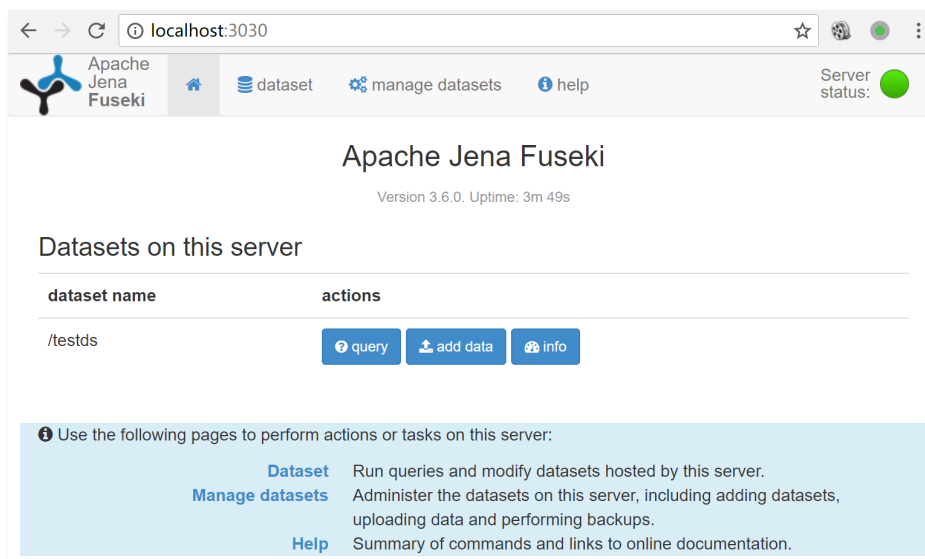


图 3-4 Jena Fuseki 的 Web 用户界面

3.1.4 生成知识图谱数据

首先生成一个音乐知识图谱 **Music** 用于实战练习。该音乐知识图谱的模式定义如图 3-5 所示，解释如下：

右侧虚线框中为图例，给出了 RDF 知识图谱中 4 种不同元素的表示：实体 RDF 资源实例，值是与实体关联的具体属性值，联系是实体与另一实体之间的某种关系，属性连接实体与某

个属性值；

m 代表命名空间前缀 <http://kg.course/music/> 的缩写；

包括 4 种实体，分别是歌曲 m:trackID、专辑 m:albumID 和歌手 m:artistID；

包括 2 种联系，分别是歌曲所属的专辑 m:track_album、歌曲由哪位歌手演唱 m:track_artist；

包括 4 种属性：分别是歌曲的名称 m:track_name、歌曲的标签 m:track_tag、专辑的名称 m:album_name 和歌手的名字 m:artist_name，这些属性均为字符串类型 xsd:string。

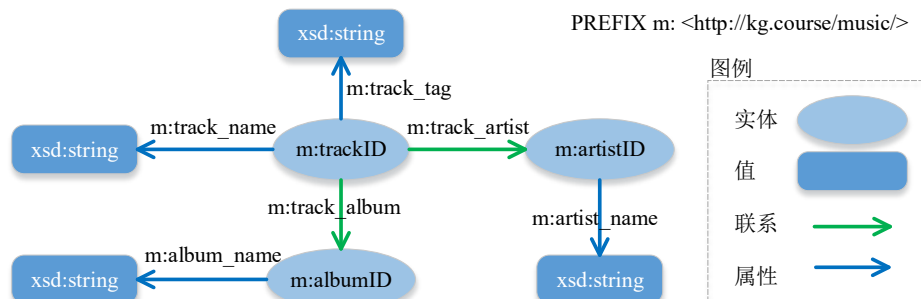


图 3-5 音乐知识图谱的模式图

根据上述模式定义生成 Music 知识图谱的步骤如下：

步骤1。使用 Python 脚本生成 Music 知识图谱，Python 脚本代码如下：

```
import random
import sys
track_name = "http://kg.course/music/track\_%05d
              <http://kg.course/music/track_name> \"track_name_%05d\" ."
track_album = "http://kg.course/music/track\_%05d
               <http://kg.course/music/track_album> <http://kg.course/music/album_%04d> ."
album_name = "<http://kg.course/music/album_%04d>
              <http://kg.course/music/album_name> \"album_name_%04d\" ."
track_artist = "http://kg.course/music/track\_%05d
                <http://kg.course/music/track_artist> <http://kg.course/music/artist_%03d> ."
artist_name = "http://kg.course/music/artist\_%03d
               <http://kg.course/music/artist_name> \"artist_name_%03d\" ."
tag_name = "<http://kg.course/music/track_%05d> <http://kg.course/music/track_tag> \"tag_name_%02d\" ."
total_sum = 1000
triples_sum = 0
triples = []
if (len(sys.argv) >= 2):
    try:
        total_sum = int(sys.argv[1])
    except:
        total_sum = 1000
for i in range(1, total_sum):
    track_str = track_name % (i, i)
    s = random.randint(1, 10)
    album_str = track_album % (i, i/s + 1)
    album_name_str = album_name % (i/10 + 1, i/10 + 1)
    t = random.randint(1, 100)
```

```

track_artist_str = track_artist % (i, i % t)
artist_name_str = artist_name % (i % t, i % t)
k = random.randint(1, 10)
tag_name_str = tag_name % (i, i % k + 1)
triples.append(track_str)
triples_sum += 1
if (total_sum <= triples_sum):
    break
triples.append(album_str)
triples_sum += 1
if (total_sum <= triples_sum):
    break
triples.append(album_name_str)
triples_sum += 1
if (total_sum <= triples_sum):
    break
triples.append(track_artist_str)
triples_sum += 1
if (total_sum <= triples_sum):
    break
triples.append(tag_name_str)
triples_sum += 1
if (total_sum <= triples_sum):
    break

filename = ("music_%d_triples.nt" % (total_sum))
with open(filename, "w+") as fd:
    fd.write("\n".join(triples))

```

执行命令：

```
python kg_music_triples.py
```

默认生成 1000 条三元组，生成的文件在当前目录，文件名为 music_1000_triples.nt；

该脚本也可传入参数，指定生成的三元组条数，例如，执行命令：

```
python kg_music_triples.py 500
```

传入参数 500，生成含有 500 条三元组的文件，文件名为 music_500_triples.nt；

步骤2。打开 music_1000_triples.nt 文件，前 5 行如下：

```

m:track_00001 m:track_name "track_name_00001" .
m:track_00001 m:track_album m:album_0001 .
m:album_0001 m:album_name "album_name_0001" .
m:track_00001 m:track_artist m:artist_001 .
m:track_00001 m:track_tag "tag_name_02" .

```

表示歌曲 m:track_00001 的名称是 track_name_00001，它所属专辑为 m:album_0001，该专辑名称为 album_name_0001。该歌曲由歌手 m:artist_001 演唱，该歌曲的标签是 tag_name_02；

步骤3。注意到上述生成数据中没有为歌手生成 m:artist_name 属性，这是为了后续练习数据更新的需要，后面我们将通过 SPARQL Update 语句为歌手添加这个属性。

3.1.5 将知识图谱装载到 Fuseki

本小节的任务是将已经生成的含有 1000 条三元组的 Music 知识图谱数据 `music_1000_triples.nt` 装载到 Fuseki 数据库。步骤如下：

步骤1。在图 3-4 所示的 Fuseki 用户界面中单击【add data】按钮，向数据集/testds 中添加数据，转到如图 3-6 所示的页面，单击【select files...】按钮，浏览到 `music_1000_triples.nt` 文件并将其选中，单击【upload now】将文件上传；

步骤2。如文件上传成功，则显示如图 3-7 所示的界面；表明成功地装载了 1000 条三元组。

提示：好奇的读者会问：上传到 Fuseki 的知识图谱三元组数据究竟存储到了什么地方？还记得启动 Fuseki 服务器时指定的命令行选项--loc=store 吗？没错！在 Fuseki 安装目录中建立的目录 `store` 正是 Fuseki 存放数据库后台文件的位置；刚刚上传成功的 Music 知识图谱就是被存储到了这些数据文件中。Fuseki 数据库后台数据文件列表如图 3-8 所示。

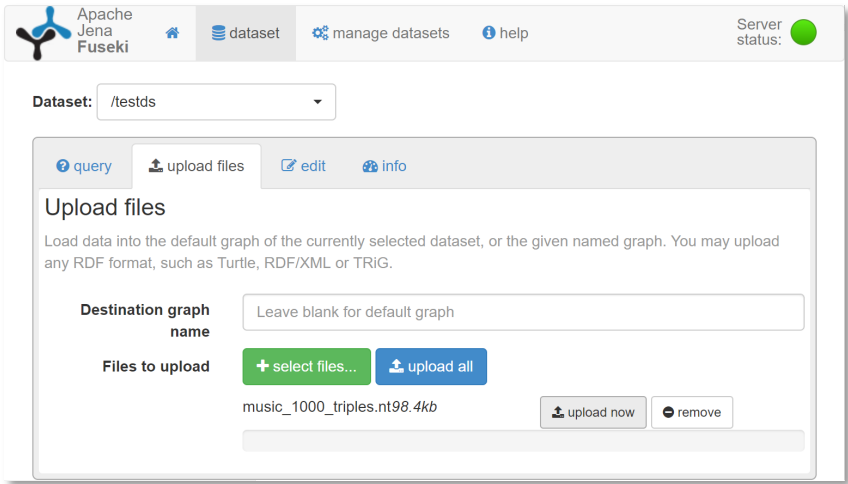


图 3-6 上传知识图谱文件到 Fuseki

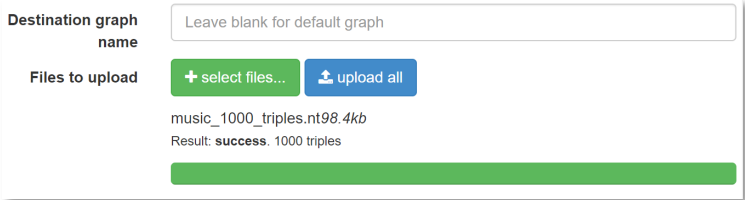


图 3-7 上传知识图谱文件成功

名称	修改日期	类型	大小
GPOS.dat	2018/1/27 17:20	DAT 文件	8,192 KB
GPOS.idn	2018/1/27 17:20	IDN 文件	8,192 KB
GSPO.dat	2018/1/27 17:20	DAT 文件	8,192 KB
GSPO.idn	2018/1/27 17:20	IDN 文件	8,192 KB
journal.jrnl	2018/1/28 10:26	JRNL 文件	0 KB
node2id.dat	2018/1/27 17:20	DAT 文件	8,192 KB
node2id.idn	2018/1/27 17:20	IDN 文件	8,192 KB
nodes.dat	2018/1/28 10:26	DAT 文件	20 KB
OSP.dat	2018/1/27 17:20	DAT 文件	8,192 KB
OSP.idn	2018/1/27 17:20	IDN 文件	8,192 KB
OSPG.dat	2018/1/27 17:20	DAT 文件	8,192 KB
OSPG.idn	2018/1/27 17:20	IDN 文件	8,192 KB
POS.dat	2018/1/27 17:20	DAT 文件	8,192 KB
POS.idn	2018/1/27 17:20	IDN 文件	8,192 KB
POSG.dat	2018/1/27 17:20	DAT 文件	8,192 KB
POSG.idn	2018/1/27 17:20	IDN 文件	8,192 KB
prefix2id.dat	2018/1/27 17:20	DAT 文件	8,192 KB
prefix2id.idn	2018/1/27 17:20	IDN 文件	8,192 KB
prefixes.dat	2018/1/27 17:20	DAT 文件	0 KB
prefixidx.dat	2018/1/27 17:20	DAT 文件	8,192 KB
prefixidx.idn	2018/1/27 17:20	IDN 文件	8,192 KB
SPO.dat	2018/1/27 17:20	DAT 文件	8,192 KB
SPO.idn	2018/1/27 17:20	IDN 文件	8,192 KB
SPOG.dat	2018/1/27 17:20	DAT 文件	8,192 KB
SPOG.idn	2018/1/27 17:20	IDN 文件	8,192 KB
tdb.lock	2018/1/27 17:20	LOCK 文件	1 KB

28 个项目



图 3-8 Fuseki 数据库后台数据文件列表

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 Music 知识图谱上的实际操作。

3.1.6 查询知识图谱

1. 热身准备

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。单击图 3-6 左上方的主页图标，返回如图 3-4 所示的页面，单击【query】按钮；或者单击上方的【dataset】数据集图标，再单击【query】查询选项卡；都会进入如图 3-9 所示的 Fuseki SPARQL 查询页面。

Dataset: /testds

query
upload files
edit
info

SPARQL query

To try out some SPARQL queries against the selected dataset, enter your query here.

EXAMPLE QUERIES

Selection of triples
Selection of classes

PREFIXES

rdf
rdfs
owl
xsd
+

SPARQL ENDPOINT

CONTENT TYPE (SELECT)

CONTENT TYPE (GRAPH)

http://localhost:3030/testds/sparql
JSON
Turtle

```

1 SELECT ?subject ?predicate ?object
2 WHERE {
3   ?subject ?predicate ?object
4 }
5 LIMIT 25

```





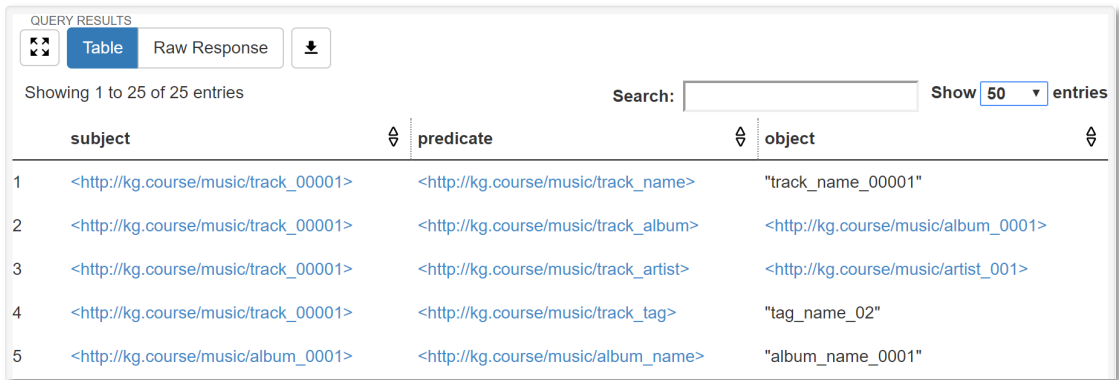




图 3-9 Fuseki SPARQL 查询页面

步骤2。在图 3-9 所示的 Fuseki SPARQL 查询界面中，最上方的“Dataset”列表指定查

询对应的数据集;“EXAMPLE QUERIES”部分有两个示例查询,分别是“Selection of triples”(选出三元组)和“Selection of classes”(选出类);“PREFIXES”部分的功能是向查询中添加前缀缩写;再下面的区域指定 SPARQL ENDPOINT(端点)和 CONTENT TYPE(内容类型),接受默认值即可;再下面是 SPARQL 查询编辑器,默认显示的就是示例查询中的“Selection of triples”,即返回 25 条三元组。

步骤3。单击 SPARQL 编辑器右上角的执行按钮,在最下方的“QUERY RESULTS”部分输出查询结果,如图 3-10 所示。



	subject	predicate	object
1	<http://kg.course/music/track_00001>	<http://kg.course/music/track_name>	"track_name_00001"
2	<http://kg.course/music/track_00001>	<http://kg.course/music/track_album>	<http://kg.course/music/album_0001>
3	<http://kg.course/music/track_00001>	<http://kg.course/music/track_artist>	<http://kg.course/music/artist_001>
4	<http://kg.course/music/track_00001>	<http://kg.course/music/track_tag>	"tag_name_02"
5	<http://kg.course/music/album_0001>	<http://kg.course/music/album_name>	"album_name_0001"

图 3-10 SPARQL 查询结果

2. 查询某一歌手演唱的所有歌曲

分析: 这个查询问题要求查找歌曲,条件是某一指定歌手演唱的所有歌曲。这涉及歌曲和歌手之间的演唱联系,即 m:track_artist;歌曲是要求查找的、未知的实体,是变量,不妨用?trackID 表示。某一歌手是指定的、已知的实体,是常量,不妨设为 m:artist_001。

构造出的 SPARQL 查询图示和查询语句,如图 3-11 所示。显然,这是一个三元组模式查询。

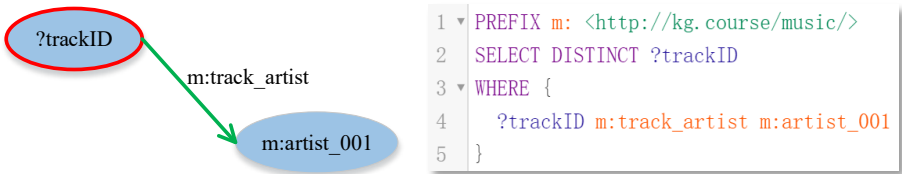
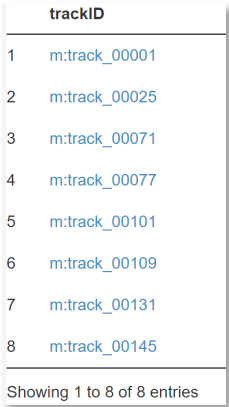


图 3-11 SPARQL 查询图示和查询语句: 查询某一歌手演唱的所有歌曲

执行该查询,返回查询结果如图 3-12 所示。可以看到歌手 m:artist_001 总共演唱了 8 首歌曲。



trackID
1 m:track_00001
2 m:track_00025
3 m:track_00071
4 m:track_00077
5 m:track_00101
6 m:track_00109
7 m:track_00131
8 m:track_00145

图 3-12 SPARQL 查询结果：歌手 m:artist_001 演唱的所有歌曲

3. 还想进一步知道歌曲名称，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歌曲 ID 对应的歌曲名称，这样的查询应该如何写呢？即查询可表达为：查询某一歌手演唱的所有歌曲 ID 和歌曲名称。

分析：只需要在上述查询基础上，为?trackID 添加 m:track_name 属性，指向变量?name，将?trackID 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图 3-13 所示。这是一个星形的基本图模式（Basic Graph Pattern，BGP）查询。

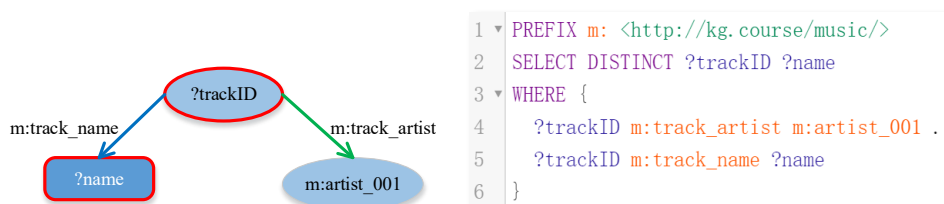


图 3-13 SPARQL 查询图示和查询语句：查询某一歌手演唱的所有歌曲 ID 和歌曲名称

执行该查询，返回查询结果如图 3-14 所示。可以看到歌手 m:artist_001 总共演唱的 8 首歌曲 ID 和相应的歌曲名称。

	trackID	name
1	m:track_00001	"track_name_00001"
2	m:track_00025	"track_name_00025"
3	m:track_00071	"track_name_00071"
4	m:track_00077	"track_name_00077"
5	m:track_00101	"track_name_00101"
6	m:track_00109	"track_name_00109"
7	m:track_00131	"track_name_00131"
8	m:track_00145	"track_name_00145"

Showing 1 to 8 of 8 entries

图 3-14 SPARQL 查询结果：歌手 m:artist_001 演唱的所有歌曲 ID 和相应的歌曲名称

4. 查询某一首歌曲名称对应的专辑信息

分析：设指定这首歌曲名称为 track_name_00001，查看 Music 知识图谱的模式图，由歌曲名称出发，沿 m:track_name 属性逆向找到歌曲 ID，再由歌曲 ID 沿 m:track_album 联系找到其所属专辑 ID；要求返回的专辑信息除了歌曲 ID、专辑 ID，还应包含专辑名称，这由专辑 ID 的属性 m:album_name 表示。因此，要用到歌曲 ID 变量?trackID、专辑 ID 变量?albumID 和专辑名称变量?name。

构造出的 SPARQL 查询图示和查询语句，如图 3-15 所示。



图 3-15 SPARQL 查询图示和查询语句：查询某一首歌曲名称对应的专辑信息

执行该查询，返回查询结果如图 3-16 所示。可以看到歌曲名称 `track_name_00001` 对应的歌曲 ID、所属的专辑 ID 和专辑名称。

trackID	albumID	name
1 m:track_00001	m:album_0001	"album_name_0001"
Showing 1 to 1 of 1 entries		

图 3-16 SPARQL 查询结果：歌曲 `track_name_00001` 对应的专辑信息

Fuseki 还支持使用中文定义 SPARQL 变量，本查询还可以写成如图 3-17 所示的形式。

```

1 PREFIX m: <http://kg.course/music/>
2 SELECT ?歌曲id ?专辑id ?专辑名
3 WHERE {
4   ?歌曲id m:track_name "track_name_00001" .
5   ?歌曲id m:track_album ?专辑id .
6   ?专辑id m:album_name ?专辑名
7 }

```

图 3-17 SPARQL 查询：用中文定义变量

如果还想在查询输出结果的专辑名前面加一些描述文字，应该怎么办呢？可以使用 SPARQL 提供的字符串操作函数 `CONCAT` 进行字符串连接操作，将描述文字连接到专辑名前面。编写查询语句，如图 3-18 所示。查询输出结果，如图 3-19 所示。

```

1 PREFIX m: <http://kg.course/music/>
2 SELECT ?歌曲id ?专辑id (CONCAT("专辑名", ":", ?专辑名) AS ?专辑信息)
3 WHERE {
4   ?歌曲id m:track_name "track_name_00001" .
5   ?歌曲id m:track_album ?专辑id .
6   ?专辑id m:album_name ?专辑名
7 }

```

图 3-18 SPARQL 查询语句：调用字符串操作函数 `CONCAT`

歌曲id	专辑id	专辑信息
1 m:track_00001	m:album_0001	"专辑名:album_name_0001"
Showing 1 to 1 of 1 entries		

图 3-19 SPARQL 查询结果：调用字符串操作函数 `CONCAT`

5. 查询某个专辑里面的所有歌曲

分析：设指定专辑名称为 `album_name_0002`，查看 Music 知识图谱的模式图，由专辑名称出发，沿 `m:album_name` 属性逆向找到专辑 ID，再由专辑 ID 沿 `m:track_album` 联系逆向找到属于该专辑的歌曲 ID。可见该查询是一个链式查询。

构造出的 SPARQL 查询图示和查询语句，如图 3-20 所示。

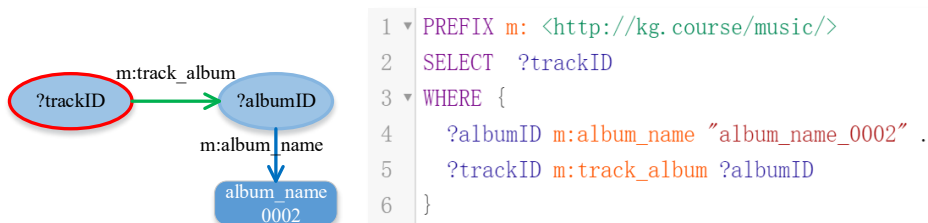


图 3-20 SPARQL 查询图示和查询语句：查询某个专辑里面的所有歌曲

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，如图 3-21 所示。

```
1 PREFIX m: <http://kg.course/music/>
2 SELECT ?trackID
3 WHERE {
4   ?albumID m:album_name "album_name_0002" .
5   ?trackID m:track_album ?albumID
6 }
7 LIMIT 2
```

图 3-21 SPARQL 查询语句：使用 LIMIT 限制查询结果条数

如果想知道该专辑中歌曲的确切数目，可使用聚合函数 COUNT 对歌曲进行计数，如图 3-22 所示。

```
1 PREFIX m: <http://kg.course/music/>
2 SELECT (COUNT(?trackID) AS ?num)
3 WHERE {
4   ?albumID m:album_name "album_name_0002" .
5   ?trackID m:track_album ?albumID
6 }
```

图 3-22 SPARQL 查询语句：使用聚合函数 COUNT 进行计数

6. 查询某一首歌是哪一个歌手的作品

分析：该查询与查询 3 “由歌手信息获得歌曲信息” 的知识图谱导航方向恰好相反。设已知歌曲名称为 track_name_00001，由 m:album_name 属性找到歌曲 ID，再由歌曲 ID 的 m:track_artist 联系找到演唱该歌曲的歌手。该查询是一个星形查询。

构造出的 SPARQL 查询图示和查询语句，如图 3-23 所示。

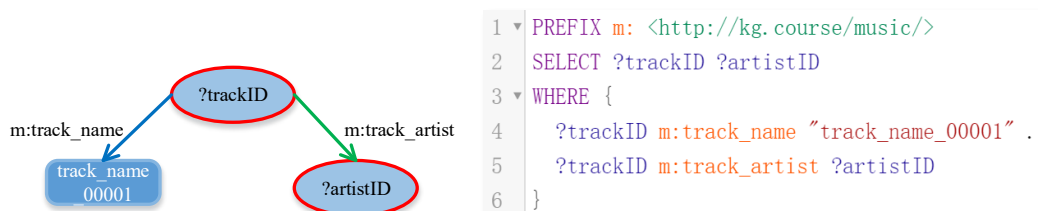


图 3-23 SPARQL 查询图示和查询语句：查询某一首歌是哪一个歌手的作品

7. 查询某一首歌属于什么歌曲标签

分析：该查询与查询 6 的模式相同，只是将“歌曲—歌手”联系换成了“歌曲—标签”联系。

构造出的 SPARQL 查询图示和查询语句，如图 3-24 所示。

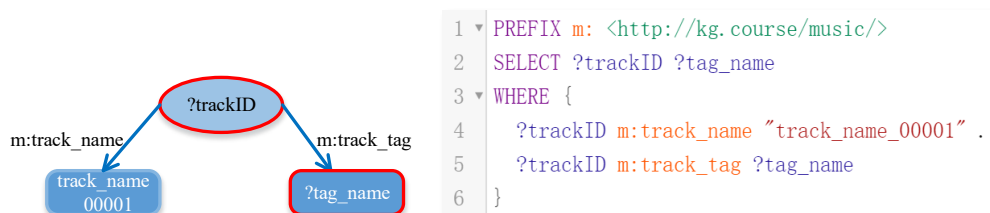


图 3-24 SPARQL 查询图示和查询语句：查询某一首歌属于什么歌曲标签

8. 查询某一歌手唱过歌曲的所有标签

分析：该查询也与查询 6、7 的模式相同，区别是将查询 7 中的歌曲名称属性 `m:track_name` 改为了“歌曲—歌手”联系 `m:track_artist`。

构造出的 SPARQL 查询图示和查询语句，如图 3-25 所示。

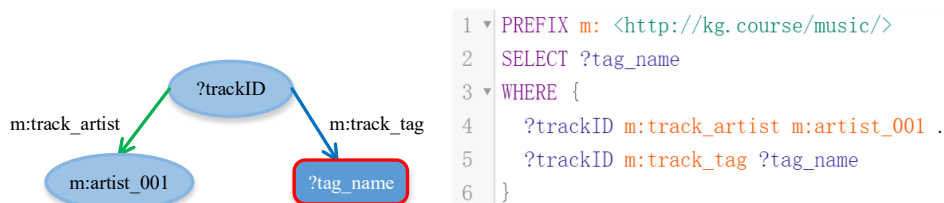


图 3-25 SPARQL 查询：查询某一歌手唱过歌曲的所有标签

但观察该查询的返回结果，发现其中有重复的歌曲标签，这并不奇怪，因为该歌手演唱的不同歌曲可能具有相同的标签。如果去掉重复呢？还记得 `DISTINCT` 关键字吗？只需要在 `SELECT` 后面加上 `DISTINCT`，即可去掉查询结果中的重复值，如图 3-26 所示。

```
1 PREFIX m: <http://kg.course/music/>
2 SELECT DISTINCT ?tag_name
3 WHERE {
4   ?trackID m:track_artist m:artist_001 .
5   ?trackID m:track_tag ?tag_name
6 }
```

图 3-26 SPARQL 查询：使用 `DISTINCT` 关键字去掉重复值

这时，另一个需求来了！用户提出要给结果按标签排序。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 ?tag_name` 子句，即可将结果按照标签的升序进行排列，如图 3-27 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。

```
1 PREFIX m: <http://kg.course/music/>
2 SELECT DISTINCT ?tag_name
3 WHERE {
4   ?trackID m:track_artist m:artist_001 .
5   ?trackID m:track_tag ?tag_name
6 }
7 ORDER BY ?tag_name
```

图 3-27 SPARQL 查询：使用 `ORDER BY` 关键字进行排序

9. 查询某几类歌曲标签中的歌曲数目

分析：首先，求歌曲数目是一个统计查询，需要用到聚合函数 `COUNT`。不妨设求出具有两类标签 `tag_name_01` 和 `tag_name_02` 的歌曲数目。一种方法是分别求出具有每类标签的所有歌曲，然后对这两类歌曲求并集操作；另一种方法是采用 `FILTER` 关键字将两类标签设定为歌曲标签的过滤条件。

构造出相应的 SPARQL 查询语句，如图 3-28 所示。

```

1 PREFIX m: <http://kg.course/music/>
2 SELECT (COUNT(?trackID) AS ?num)
3 WHERE {
4   { ?trackID m:track_tag "tag_name_01" . }
5   UNION
6   { ?trackID m:track_tag "tag_name_02" . }
7 }

```

图 3-28 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，如图 3-29 所示。

```

1 PREFIX m: <http://kg.course/music/>
2 SELECT (COUNT(?trackID) AS ?num)
3 WHERE {
4   ?trackID m:track_tag ?tag_name
5   FILTER (?tag_name = "tag_name_01" || ?tag_name = "tag_name_02")
6 }

```

图 3-29 SPARQL 查询：使用 FILTER 关键字进行过滤

10. 询问是否存在含有某字符串的歌曲名

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，如图 3-30 所示。

```

1 PREFIX m: <http://kg.course/music/>
2 ASK {
3   ?trackID m:track_name ?track_name
4   FILTER regex(?track_name, "008")
5 }

```

图 3-30 SPARQL 查询语句：询问是否存在含有某字符串的歌曲名
ASK 查询的结果只有 True 或者 False，例如上述查询的返回结果，如图 3-31 所示。

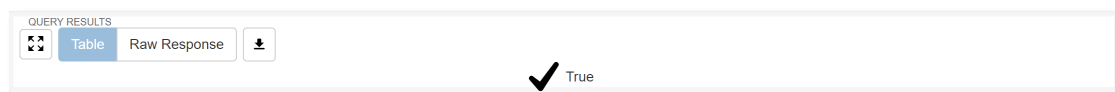


图 3-31 SPARQL 查询：ASK 查询的返回结果

3.1.7 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。

1. 给歌手 ID 新增属性歌手名字

分析：请回顾图 3-5 给出的音乐知识图谱模式图，其中歌手 ID 的属性 m:artist_name 表示该歌手的名字，但是我们生成音乐知识图谱数据时并没有为歌手生成名字，即没有添加 m:artist_name 属性值。下面通过 INSERT 关键字向音乐知识图谱中插入 3 位歌手的名字。

需要注意的是，在执行 SPARQL 更新语句前，需要将 Fuseki 用户界面中的 SPARQL Endpoint 由 http://localhost:3030/testds/sparql 改为 http://localhost:3030/testds/update，如图 3-32 所示。

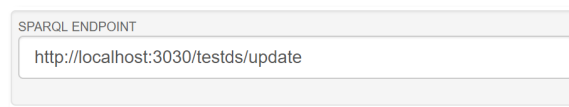


图 3-32 执行 SPARQL 更新语句前更改 SPARQL Endpoint 值

给 ID 为 m:artist_001、m:artist_002 和 m:artist_003 的歌手分别添加名字 artist_name_001、artist_name_002 和 artist_name_003，构造 SPARQL 更新语句，如图 3-33 所示。

```
1 PREFIX m: <http://kg.course/music/>
2 INSERT DATA {
3   m:artist_001 m:artist_name "artist_name_001" .
4   m:artist_002 m:artist_name "artist_name_002" .
5   m:artist_003 m:artist_name "artist_name_003" .
6 }
```

图 3-33 SPARQL 更新语句：给歌手 id 新增属性歌手名字

执行该更新操作，如返回的是一段 HTML 文本，其中含有 Success Update succeeded 字样，则表明更新操作执行成功。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。（注意，执行查询前需要再将 SPARQL Endpoint 改回 http://localhost:3030/testds/sparql）

2. 删除增加的属性歌手名字

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。

删除歌手 m:artist_002 的名字属性，构造 SPARQL 更新语句，如图 3-34 所示。（注意，执行删除操作仍然要用 http://localhost:3030/testds/update 作为 SPARQL Endpoint）

```
1 PREFIX m: <http://kg.course/music/>
2 DELETE {
3   m:artist_002 m:artist_name ?x .
4 }
5 WHERE {
6   m:artist_002 m:artist_name ?x .
7 }
```

图 3-34 SPARQL 更新语句：删除增加的属性歌手名字

与执行 INSERT 操作类似，如成功执行 DELETE 操作，将返回一段含有 Success Update succeeded 字样的 HTML 文本。可以执行一个查询，以验证符合条件的三元组 m:artist_002 m:artist_name ?x 已经被删除，该查询的编写和执行也留给读者作为练习。