

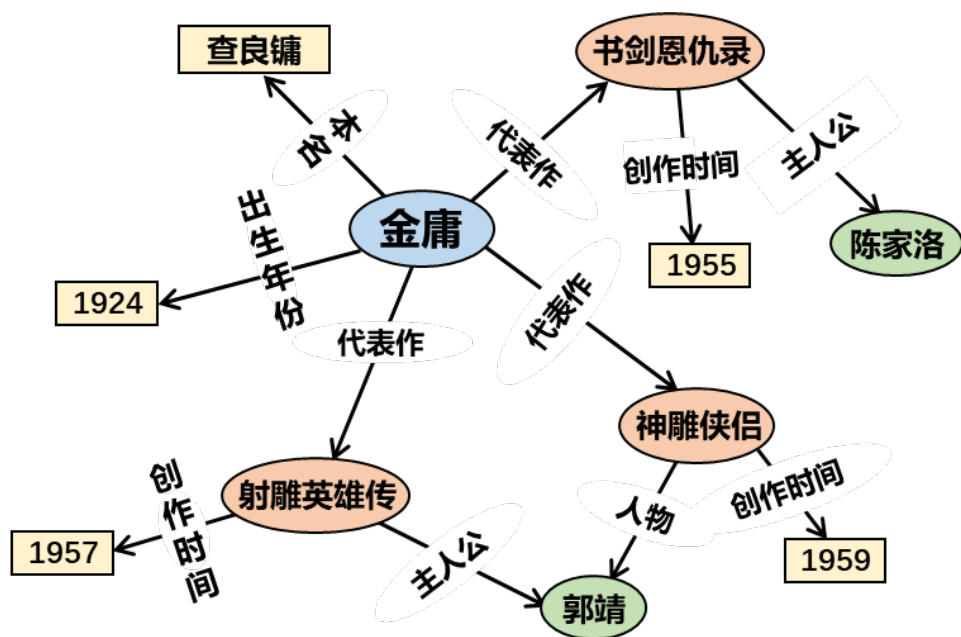
第十讲 知识存储与检索

冯骁骋

社会计算与信息检索研究中心

哈工大计算学部

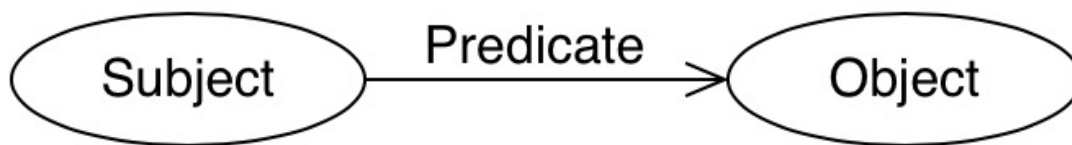
- 知识图谱是一种有向图结构，描述了现实世界中存在的实体、时间或者概念以及它们之间的关系。



- 实体：**金庸、神雕侠侣、1924等
- 实体类型：**作者、书籍、时间、角色等
- 属性：**书籍有创作时间、主人公等属性；作者有出生年份和代表作等属性
- 关系：**作者和书籍之间的代表作关系，书籍和角色之间的主人公关系等

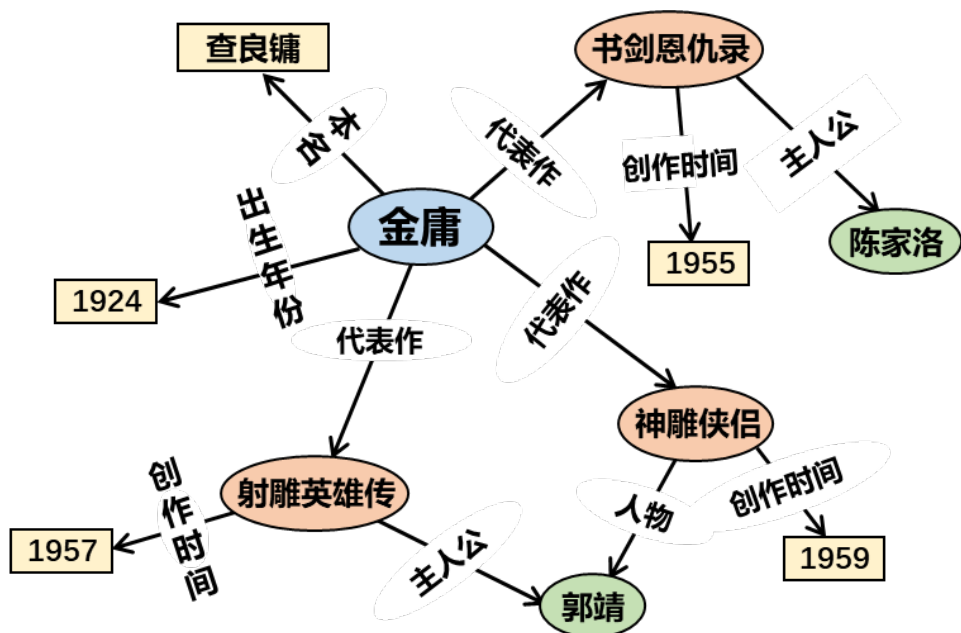
知识图谱中的知识表示

- ▶ 知识图谱中的知识是通过RDF的结构进行表示的，其基本构成单元是事实，每个事实被表示为一个形如<subject, predicate, object>的三元组。
 - ▶ subject: 主语，其取值通常是实体、时间或概念
 - ▶ predicate: 谓语，其取值通常是关系或者属性
 - ▶ object: 宾语，取值既可以是实体、事件、概念，也可以是普通的值（如数字、字符串等）



知识图谱中的知识表示

- 知识图谱中的知识是通过RDF的结构进行表示的，其基本构成单元是事实，每个事实被表示为一个形如<subject, predicate, object>的三元组。



RDF Data

<S, P, O>

金庸 代表作 书剑恩仇录

金庸 代表作 射雕英雄传

金庸 代表作 神雕侠侣

神雕侠侣 人物 郭靖

神雕侠侣 创作时间 1959

...

知识图谱的数据量巨大

- ▶ DBpedia包含近8000万三元组
- ▶ YAGO有超过1.2亿三元组
- ▶ Wikidata有近4.1亿三元组
- ▶ Freebase有超过30亿三元组

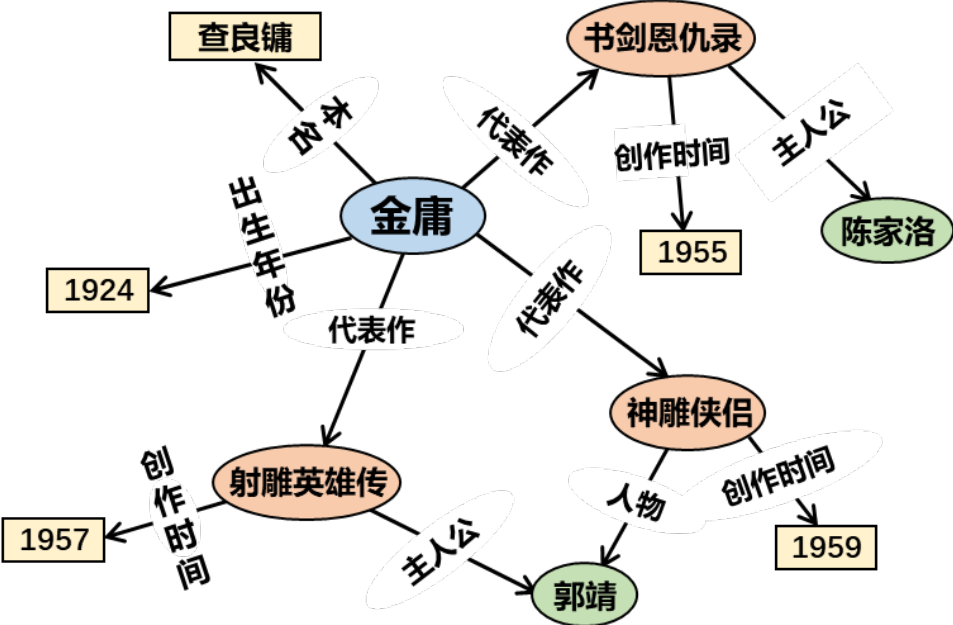
如何有效的存储与检索？

- ▶ **知识图谱的存储**
 - ▶ 基于表结构的存储
 - ▶ 基于图结构的存储
- ▶ 知识图谱的查询
 - ▶ SQL查询语言
 - ▶ SPASQL查询语言

- ▶ 按照存储方式的不同，知识图谱的存储可以分为基于表结构的存储和基于图结构的存储。
 - ▶ 基于表结构的存储：利用二维的数据表对知识图谱中的数据进行存储
 - ▶ 三元组表、类型表、关系数据库
 - ▶ 基于图结构的存储：利用图的方式对知识图谱中的数据进行存储
 - ▶ 图数据库

基于表结构的存储：三元组表

- 知识图谱中的事实是一个个的三元组，一种最简单直接的存储方式是设计一张三元组表用于存储知识图谱中的所有的事实。



S	P	O
金庸	代表作	书剑恩仇录
金庸	代表作	射雕英雄传
金庸	代表作	神雕侠侣
神雕侠侣	人物	郭靖
神雕侠侣	创作时间	1959
...

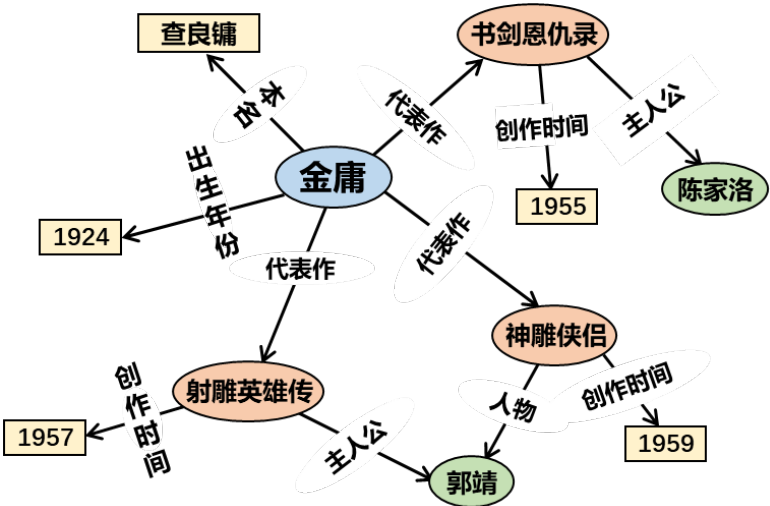
基于表结构的存储：三元组表

- ▶ 优点：简单直接、易于理解
- ▶ 缺点：
 - ▶ 整个知识图谱都存储在一张表中，导致单表的**规模太大**。对大表进行查询、插入、删除、修改等操作的开销很大，这将导致知识图谱的实用性大打折扣
 - ▶ 复杂查询在这种存储结构上的**开销巨大**。由于数据表只包括三个字段，因此复杂的查询只能拆分成若干简单查询的复合操作，大大降低了查询的效率。例如，查询“神雕侠侣的人物和创作时间”需要拆分成“神雕侠侣的人物”和“神雕侠侣的创作时间”

S	P	O
神雕侠侣	人物	郭靖
神雕侠侣	创作时间	1959

基于表结构的存储：类型表

- 为每种类型构建一张表，同一类型的实例放在同一表中
- 表中的每一列表示该类实体的一个属性，每一行存储该类实体的一个实例



文学作品表

主体对象	创作时间	主人公	人物
书剑恩仇录	1955	陈家洛	
射雕英雄传	1957	郭靖	
神雕侠侣	1959		郭靖

作者表

主体对象	代表作	本名	出生年月
金庸	神雕侠侣	查良镛	1924
金庸	射雕英雄传	查良镛	1924

类型表的不足

- 大量的数据列为空值。通常知识图谱中并非每个实体在所有属性或关系上都有值，这种存储方式会导致表中存在大量的空值

文学作品表

主体对象	创作时间	主人公	人物
书剑恩仇录	1955	陈家洛	
射雕英雄传	1957	郭靖	
神雕侠侣	1959		郭靖

类型表的不足

- 大量数据字段的冗余存储。假设知识图谱中既有“武侠小说”也有“爱情小说”，那么同属于这两个类别的实例将会被同时储存在这两个表里，其中它们的共有的属性会被重复存储

武侠小说表

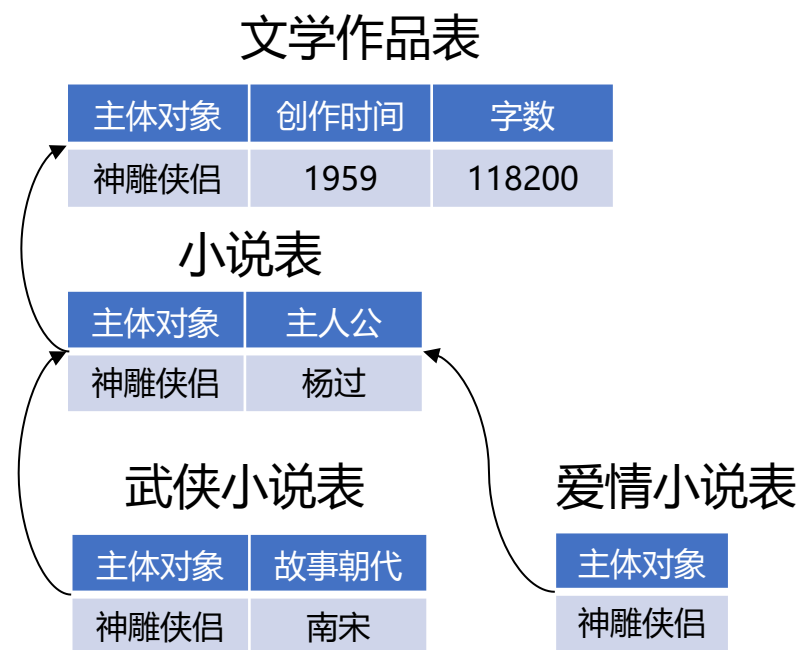
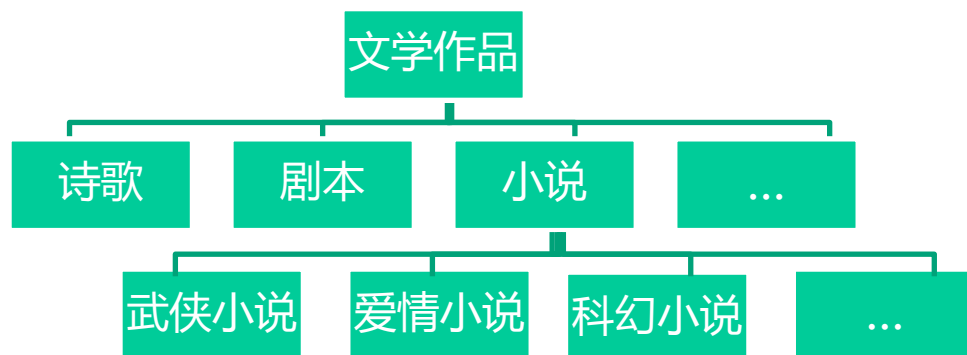
主体对象	创作时间	主人公	字数	故事朝代
神雕侠侣	1959	杨过	118200	南宋

爱情小说表

主体对象	创作时间	主人公	字数
神雕侠侣	1959	杨过	118200

考虑层级关系的类型表

- 一种有效的解决办法是，构建数据表时，将知识图谱的**类别体系**考虑进来。具体来说，每个类型的数据表只记录属于该类型的特有属性，不同类别的公共属性保存在上一级类型对应的数据表中，下级表继承上级表的所有属性。



- ▶ 类型表克服了三元组表面临的单表过大和结构简单的问题，但是也有明显的不足之处：
 - ▶ 由于数据表是和具体类型对应的，不同的数据表具有不同的结构，因此在查询之前必须知道目标对象的类型才能确定查找的数据表。
 - ▶ 当查询涉及到不同类型的实体时，需要进行多表的连接，这一操作开销较大，限制了知识图谱对复杂查询的处理能力。
 - ▶ 知识图谱通常包含丰富的实体类型，因此需要创建大量的数据表，并且这些数据表之间又具有复杂的关系，这为数据的管理增加了很大的难度。

关系数据库基本概念

- ▶ 关系数据库以二维结构对数据进行组织和存储。
 - ▶ **属性(attribute)**: 表中每一列称为一个属性（也称字段），用来描述实体集的某个特征。某个特征都有自己的取值范围，称为域。
 - ▶ **元组(tuple)**: 表中的每一行由一个实体的相关属性取值构成，称为元组（也称记录），它相对完整地描述了一个实体。元组中的一个属性值称为分量。

学号	姓名	性别	出生日期	身份证号
1001	张伟	男	1995-09-22	310103199509222317
1002	王芳	女	1996-01-04	410104199511072437
1003	李强	男	1995-10-13	510105199604082223

关系数据库基本概念

- ▶ 上述二维表格有以下限制：
 - ▶ 每一属性必须是基本的、不能再拆分的数据类型，如整型、实型、字符型等。结构或数组不能作为属性的类型。属性的所有值必须是同类型、同语义的。

姓名	籍贯	
	省	市
张伟	河北	张家口
王芳	浙江	杭州
李强	江苏	苏州



姓名	省	市
张伟	河北	张家口
王芳	浙江	杭州
李强	江苏	苏州

关系数据库基本概念：候选键

- ▶ **候选键**：能够唯一标识元组的最小的属性集合。
 - ▶ 唯一性：候选键在整个表的范围内必须具有唯一的值，不同元组不能具有相同的候选键值。
 - ▶ 最小性：候选键所包括的属性必须都是必不可少的，缺少其中的任何一个都不再具备唯一性。

学号	姓名	性别	出生日期	身份证号
1001	张伟	男	1995-09-22	310103199509222317
1002	王芳	女	1996-01-04	410104199511072437
1003	李强	男	1995-10-13	510105199604082223

Q：候选键是什么？

关系数据库基本概念：主键

- ▶ **主键**：一个数据表可以包含多个候选键，从中任意选取一个作为主键。虽然理论上所有的候选键都可以作为主键，但是实际操作中一般选择单属性组成的候选键作为主键。
 - ▶ 在下表中有三个候选键：学号、身份证号
 - ▶ 身份证号太长，使用不方便
 - ▶ 学号简洁，使用方便，在实际生活中也被用来唯一区分学生

学号	姓名	性别	出生日期	身份证号
1001	张伟	男	1995-09-22	310103199509222317
1002	王芳	女	1996-01-04	410104199511072437
1003	李强	男	1995-10-13	510105199604082223

关系数据库基本概念：外键

- ▶ **外键**：如果数据表中的某个属性或属性组是其它表的候选键，那么称该属性或属性组是当前表的外键。外键能够保证不同数据表之间数据的一致性。
 - ▶ 班号是班级表的候选键，所以班号是学生表的外键。

学生表

学号	姓名	性别	出生日期	班号
1001	张伟	男	1995-09-22	101
1002	王芳	女	1996-01-04	102
1003	李强	男	1995-10-13	103

班级表

班号	班名	专业
101	数学01	应用数学
102	物理01	理论物理
103	化学01	材料化学

关系数据库操作语言

- ▶ 关系数据库的基本操作语言是SQL (Structured Query Language)。
- ▶ SQL语言以简洁的语法支持关系数据库的各类操作（插入/修改/删除/查询）。
- ▶ SQL语言独立于关系数据库本身，独立于使用的机器、网络和操作系统。
- ▶ 如果选择关系数据库作为知识图谱的存储引擎，那么对知识图谱的所有操作都需要转换为SQL语句才能执行

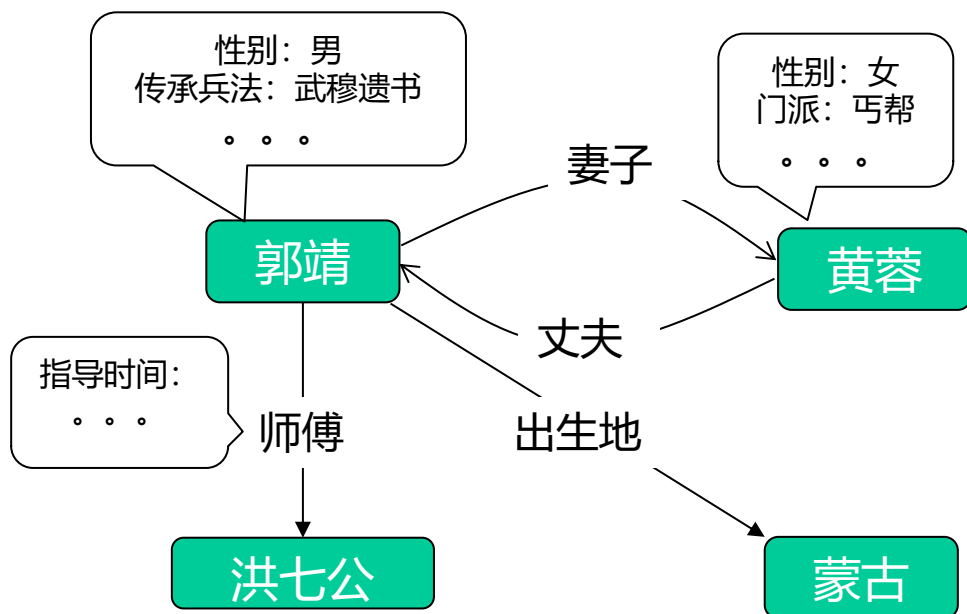
常见关系数据库存储系统

- ▶ **DB2:** DB2是IBM公司于1983年推出的数据库管理系统，能够运行于各种不同的操作系统平台上，如Windows、Linux、VMS、OS/2等。该数据库的特色之处包括支持面向对象的编程、支持多媒体应用、支持分布式数据库的访问等。
- ▶ **Oracle:** Oracle是甲骨文公司于1979年推出的关系数据库管理系统，是目前最流行的数据库之一。该数据库遵循标准SQL语言，支持多种数据类型。和DB2相同，Oracle同样具有跨平台的特点，可以运行于不同的操作系统之上。
- ▶ **MySQL:** MySQL 也是一个自由软件系统，最早是由瑞典MySQL AB公司开发的小型关系数据库管理系统，在2008年被SUN公司收购。该系统由于具有体积小、速度快、跨平台、免费开源等特点，是中小型网站最为青睐的数据库。

- ▶ 知识图谱的存储
 - ▶ 基于表结构的存储
 - ▶ 基于图结构的存储
- ▶ 知识图谱的查询
 - ▶ SQL查询语言
 - ▶ SPASQL查询语言

基于图结构的存储模型

- ▶ 将实体看作节点，关系看作带有标签的边，那么知识图谱的数据很自然地满足图模型结构。



- ▶ 用节点表示实体：郭靖、黄蓉、蒙古等
- ▶ 用边表示实体间的关系：郭靖和洪七公之间的师徒关系等
- ▶ 节点可以定义属性：黄蓉性别女、门派丐帮等
- ▶ 边上也可以定义属性：洪七公对郭靖的指导时间等

- ▶ 图数据库基于有向图，其理论基础是图论。节点、边和属性是图数据库的核心概念。
 - ▶ **节点**：节点用于表示实体、事件等对象，可以类比于关系数据库中的记录或数据表中的行数据。例如人物、地点、电影等都可以作为图中的节点。
 - ▶ **边**：边是指图中连接节点的有向线条，用于表示不同节点之间的关系。例如人物节点之间的夫妻关系、同事关系等都可以作为图中的边。
 - ▶ **属性**：属性用于描述节点或者边的特性。例如人物的姓名、夫妻关系的起止时间等都是属性。

常见的图数据库存储系统

► Neo4j

- 开源的图数据库系统，它将结构化的数据存储存储在图上而不是表中。Neo4j基于Java实现，它是一个具备完全事务特性的高性能数据库，具有成熟数据库的所有特性。Neo4j是一个本地数据库（又称基于文件的数据库），这意味着不需要启动数据库服务器，应用程序不用通过网络访问数据库服务，而是直接在本地对其进行操作，因此访问速度快。因其开源、高性能、轻量级等优势，Neo4j受到越来越多的关注。

► InfiniteGraph:

- 基于Java语言开发的分布式图数据库系统。和MySQL等传统关系数据库类似，InfiniteGraph需要作为服务项目进行安装，应用程序只能通过访问数据库服务对数据库进行操作。InfiniteGraph借鉴了面向对象的概念，将图中的每个节点及每条边都看做一个对象。具体地，所有的节点都继承自基类BaseVertex，所有的边则都继承自基类BaseEdge。

► 还有OrientDB, InfoGrid, HyperGraphDB等

- ▶ 知识图谱的存储
 - ▶ 基于表结构的存储
 - ▶ 基于图结构的存储
- ▶ 知识图谱的查询
 - ▶ SQL查询语言
 - ▶ SPASQL查询语言

- ▶ 知识图谱的知识实际上是通过数据库系统进行存储的, 大部分数据库系统通过形式化的查询语言为用户提供访问数据的接口
 - ▶ 关系数据库: 标准查询语言SQL
 - ▶ 图数据库: 标准查询语言SPARQL
 - ▶ SPARQL Protocol and RDF Query Language,是为RDF开发的一种查询语言和数据获取协议

- ▶ SQL (Structured Query Language, 结构化查询语言) 是介于关系代数和元组演算之间的一种语言, 用于管理关系数据库, 目前已经成为关系数据库的标准语言。
- ▶ SQL的主要功能包括对数据的**插入**、**修改**、**删除**、**查询**四种操作。
- ▶ 接下来我们将以下表为示例展示SQL语言的各种操作。

学号	姓名	性别	年龄	班号
1001	张伟	男	26	101
1002	王芳	女	25	102
1003	李强	男	26	103

SQL语言：数据插入

- ▶ **数据插入**是指向数据表中添加新的数据记录，SQL通过INSERT语句完成该功能。其基本语法为：

INSERT

INTO 表名 [(列 1, 列 2, ...)]

VALUES (值 1, 值 2, ...) [(值 1, 值 2, ...), ...]

- ▶ 该语句的功能是将新元组插入指定表中。
- ▶ INTO后的中括号为可选内容，代表插入数据的列。若指明则 (值 1, 值 2 ...) 分别对应着 (列 1, 列 2 ...)。否则 (值 1, 值 2 ...) 默认对应着数据表中原本的 (第一列, 第二列 ...)。
- ▶ VALUES后的中括号同样为可选内容，可以加入多个值对，相当于一次插入多行数据到表中。

SQL语言：数据插入

- ▶ 例：将一个新学生元组 (学号: 1004, 姓名: 郭栋; 性别: 男; 年龄: 23, 班号: 101) 插入到学生表中

INSERT

INTO 学生

VALUES ('1004', '郭栋', '男', 23, '101')

INSERT

INTO 学生 (学号, 姓名, 性别, 年龄, 班号)

VALUES ('1004', '郭栋', '男', 23, '101')

学号	姓名	性别	年龄	班号
1001	张伟	男	26	101
1002	王芳	女	25	102
1003	李强	男	26	103
1004	郭栋	男	23	101

SQL语言：数据修改

- ▶ **数据修改**是指修改数据表中已有记录某些列的值，SQL通过UPDATE语句完成该功能。其基本语法为：

UPDATE 表名

SET 列 1=值 1, 列 2=值 2, ...

WHERE 条件

- ▶ SET语句后的值 1, 值 2, ... 可以是表达式的形式
- ▶ WHERE语句中的“条件”用于指明需要修改的数据记录。如果省略，则表示要修改表中的所有元组。

SQL语言：数据插入

- ▶ 例：将学号为1001的学生的年龄改为24岁。

UPDATE 学生

SET 年龄=24

WHERE 学号='1001'

学号	姓名	性别	年龄	班号
1001	张伟	男	26	101
1002	王芳	女	25	102
1003	李强	男	26	103



学号	姓名	性别	年龄	班号
1001	张伟	男	24	101
1002	王芳	女	25	102
1003	李强	男	26	103

SQL语言：数据删除

- ▶ **数据删除**是指从数据表中删除指定的数据记录，SQL通过DELETE语句完成该功能。其基本语法为：

DELETE

FROM 表名

WHERE 条件

- ▶ WHERE语句中的“条件”用于指明需要删除的数据记录。如果省略，则表示要清空整张表。

SQL语言：数据删除

- ▶ 例：删除学号为1001或班号为103的学生记录。

DELETE

FROM 学生

WHERE 学号='1001' or 班号='103'

学号	姓名	性别	年龄	班号
1001	张伟	男	26	101
1002	王芳	女	25	102
1003	李强	男	26	103



学号	姓名	性别	年龄	班号
1002	王芳	女	25	102

SQL语言：数据查询

- ▶ 数据查询是指从数据表中选取满足条件的数据，查询结果存储在一个临时的结果表中。SQL通过SELECT语句完成该功能，其基本语法为：

SELECT 列 1, 列 2, ...

FROM 表名

WHERE 条件

- ▶ 主语句**SELECT-FROM-WHERE**的含义是：根据WHERE子句的条件表达式，从FROM子句指定的基本表或视图找出满足条件的元组，再按SELECT子句中的目标列表表达式，选出元组中的属性值形成结果表。

SQL语言：数据查询

- ▶ 如果要查询指定数据表所有列的数据，可以通过如下语句进行查询：

SELECT *

FROM 表名

WHERE 条件

- ▶ 例：查询所有年龄为26的同学的姓名。

SELECT 姓名

FROM 学生

WHERE 年龄=26

学号	姓名	性别	年龄	班号
1001	张伟	男	26	101
1002	王芳	女	25	102
1003	李强	男	26	103



姓名
张伟
李强

- ▶ 知识图谱的存储
 - ▶ 基于表结构的存储
 - ▶ 基于图结构的存储
- ▶ 知识图谱的查询
 - ▶ SQL查询语言
 - ▶ SPASQL查询语言

SPARQL语言: RDF回顾

- ▶ RDF (Resource Description Framework, 资源描述框架) 是一种资源描述语言, 其核心思想是利用Web标识符 (URI) 来标识事物, 通过指定的属性和相应的值描述资源的性质或资源之间的关系。

```
# Default Graph
```

```
@prefix ns: <http://example.org/ns#> .
```

```
ns: 金庸 ns: 本名 ns: 查良镛
```

- ▶ 上面是只包含一个三元组的RDF图实例。

- ▶ SPARQL是Simple Protocol and RDF Query Language的缩写，是由W3C为RDF数据开发的一种查询语言和数据获取协议，是被图数据库广泛支持的查询语言。和SQL类似，SPARQL也是一种结构化的查询语言，用于对数据的获取与管理。
 - ▶ 数据操纵（插入、删除、更新）
 - ▶ 数据查询

- ▶ **数据插入：** 将新的三元组插入到已有的RDF图中。 SPARQL通过 INSERT DATA语句完成该功能，其基本语法为：

INSERT DATA 三元组数据

- ▶ 三元组数据可以是多条三元组，不同的三元组通过 '.' 分隔，用 ';' 可以连续插入与前一个三元组的头实体相同的三元组。
- ▶ 如果待插入的三元组在RDF图中已经存在，那么系统就会忽略该三元组。

- ▶ 例：向右图中插入如下三元组。

ns: 金庸 ns: 代表作 ns: 射雕英雄传

ns: 射雕英雄传 ns: 创作时间 ns: 1957

ns: 射雕英雄传 ns: 主人公 ns: 郭靖

- ▶ 对应的SPARQL语句为：

```
prefix ns: <http://example.org/ns#>
```

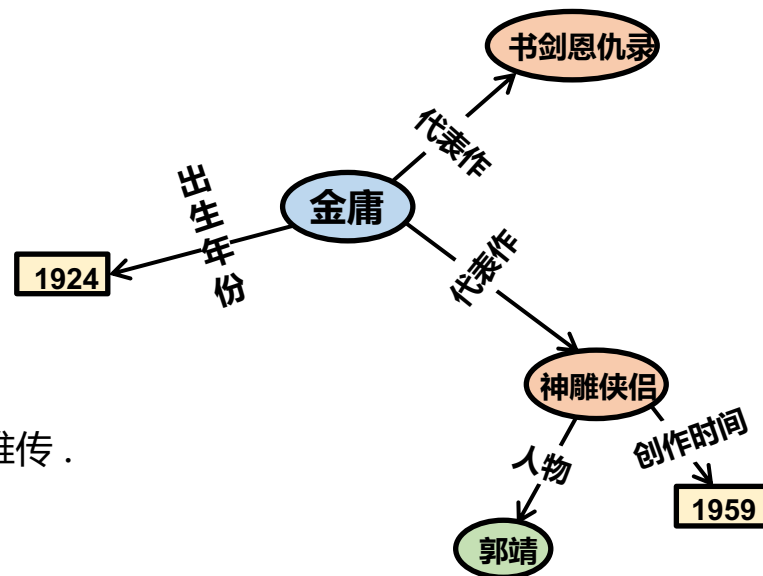
```
INSERT DATA {
```

```
  ns: 金庸      ns: 代表作  ns: 射雕英雄传 .
```

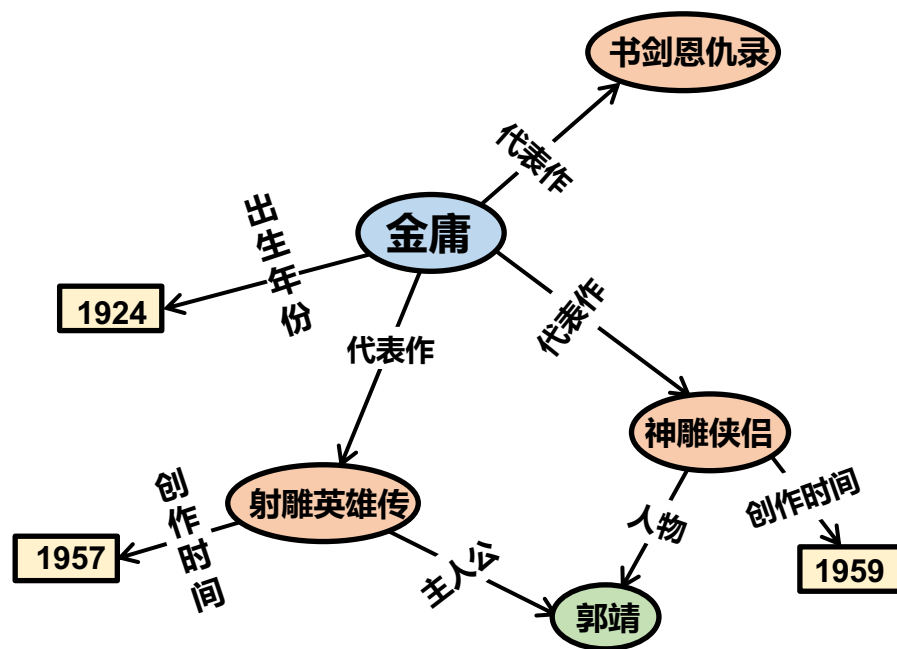
```
  ns: 射雕英雄传 ns: 创作时间 ns: 1957 ;
```

```
      ns: 主人公  ns: 郭靖 .
```

```
}
```



► 插入结果示例



@prefix ns: <http://example.org/ns#> .

ns: 金庸 ns: 代表作 ns: 书剑恩仇录

ns: 金庸 ns: 代表作 ns: 神雕侠侣

ns: 金庸 ns: 出生年份 ns: 1924

ns: 神雕侠侣 ns: 任务 ns: 郭靖

ns: 神雕侠侣 ns: 创作时间 ns: 1959

ns: 金庸 ns: 代表作 ns: 射雕英雄传

ns: 射雕英雄传 ns: 创作时间 ns: 1957

ns: 射雕英雄传 ns: 主人公 ns: 郭靖

► 接下来我们将以上图为示例展示SPARQL语言的其他操作。

- ▶ **数据删除**：从RDF图中删除一些三元组。 SPARQL通过DELETE DATA语句完成该功能，其基本语法为：

DELETE DATA 三元组数据

- ▶ 三元组数据可以是多条三元组，不同的三元组通过‘.’分隔。
- ▶ 对于给定的每个三元组，如果其在RDF图中，则将其从图中删除，否则忽略该三元组。

- 例：从之前的图中删除三元组 (ns: 射雕英雄传 ns: 主人公 ns: 郭靖) 的SPARQL语句为：

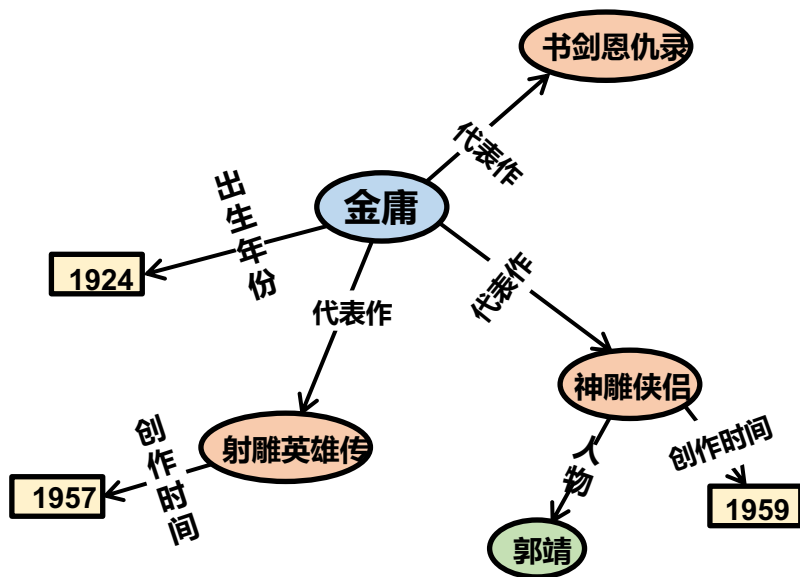
```
prefix ns: <http://example.org/ns#>
```

```
DELETE DATA {
```

```
  ns: 射雕英雄传 ns: 主人公 ns: 郭靖 .
```

```
}
```

- 删除后的图谱以及RDF数据为：



```
@prefix ns: <http://example.org/ns#> .
```

```
ns: 金庸 ns: 代表作 ns: 书剑恩仇录
```

```
ns: 金庸 ns: 代表作 ns: 神雕侠侣
```

```
ns: 金庸 ns: 出生年份 ns: 1924
```

```
ns: 神雕侠侣 ns: 任务 ns: 郭靖
```

```
ns: 神雕侠侣 ns: 创作时间 ns: 1959
```

```
ns: 金庸 ns: 代表作 ns: 射雕英雄传
```

```
ns: 射雕英雄传 ns: 创作时间 ns: 1957
```

数据删除

- 例：从之前的图中删除“射雕英雄传”对应的节点的SPARQL语句为：

prefix ns: <http://example.org/ns#>

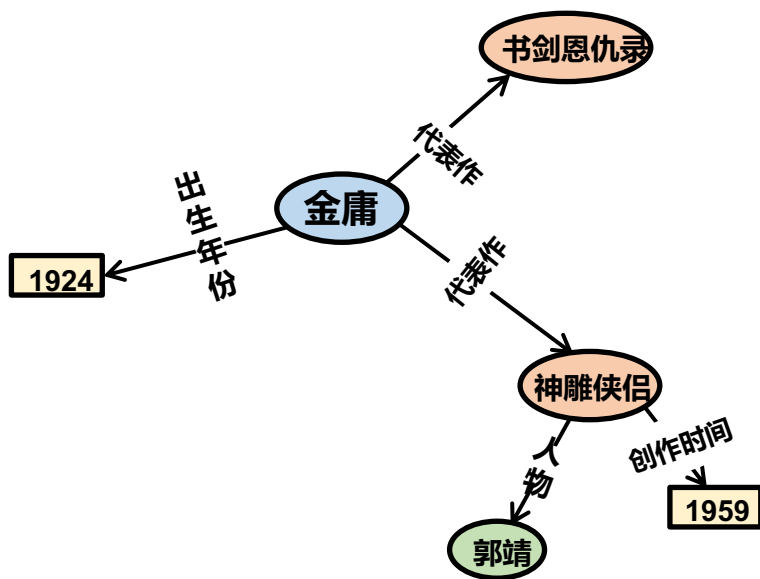
DELETE DATA {

ns: 射雕英雄传 ?p ?o .

?s ?p ns: 射雕英雄传 .

}

- 删除后的图谱以及RDF数据为：



@prefix ns: <http://example.org/ns#> .

ns: 金庸 ns: 代表作 ns: 书剑恩仇录

ns: 金庸 ns: 代表作 ns: 神雕侠侣

ns: 金庸 ns: 出生年份 ns: 1924

ns: 神雕侠侣 ns: 任务 ns: 郭靖

ns: 神雕侠侣 ns: 创作时间 ns: 1959

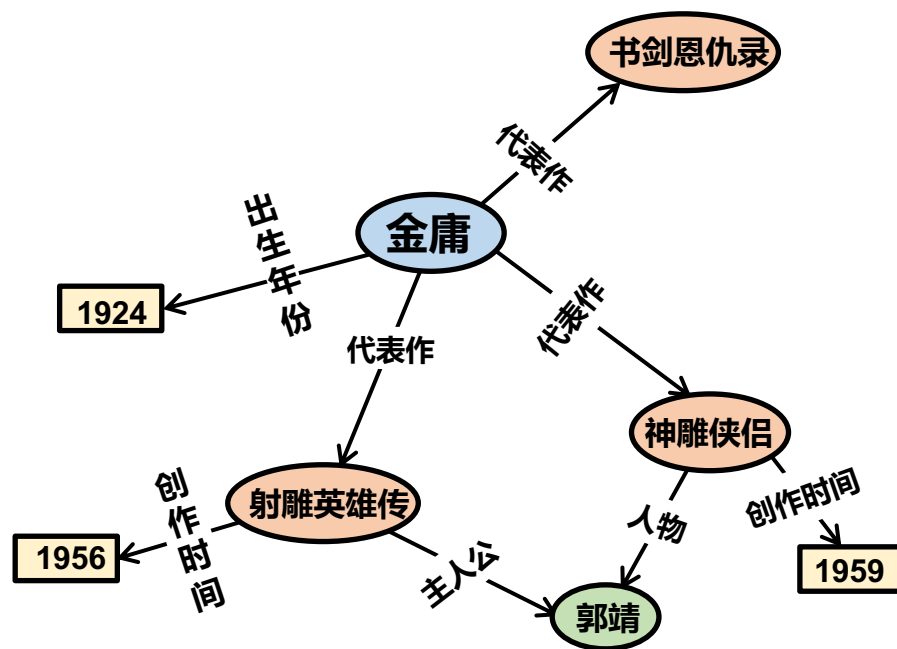
- ▶ **数据更新**：更新RDF图中指定三元组的值。和SQL不同，SPARQL没有定义UPDATE操作，也就是说SPARQL语言没有直接更新已有数据的方法。但是，可以通过组合INSERT DATA语句和DELETE DATA语句来实现该功能。
 - ▶ DELETE DATA: 删除待修改的三元组；
 - ▶ INSERT DATA: 插入修改后的三元组。
- ▶ 例：将之前图中三元组 (ns: 射雕英雄传 ns: 创作时间 ns: 1957) 的值 “1957” 改为 “1956” ，其SPARQL代码为

prefix ns: <http://example.org/ns#>

DELETE DATA {ns: 射雕英雄传 ns: 创作时间 ns: 1957 . }

INSERT DATA {ns: 射雕英雄传 ns: 创作时间 ns: 1956 . }

► 更新结果示例



@prefix ns: <http://example.org/ns#> .

ns: 金庸 ns: 代表作 ns: 书剑恩仇录

ns: 金庸 ns: 代表作 ns: 神雕侠侣

ns: 金庸 ns: 出生年份 ns: 1924

ns: 神雕侠侣 ns: 任务 ns: 郭靖

ns: 神雕侠侣 ns: 创作时间 ns: 1959

ns: 金庸 ns: 代表作 ns: 射雕英雄传

ns: 射雕英雄传 ns: 主人公 ns: 郭靖

ns: 射雕英雄传 ns: 创作时间 ns: 1956

- ▶ SPARQL提供了丰富的数据查询功能，包括四种形式：
 - ▶ **SELECT**: 最为常用的查询语句，其功能和SQL中的SELECT语句类似，从知识图谱中获取满足条件的数据。
 - ▶ **ASK**: 用于测试知识图谱中是否存在满足给定条件的数据，如果存在则返回 “yes”，否则返回 “no”，该查询不会返回具体的匹配数据。
 - ▶ **DESCRIBE**: 用于查询和指定资源相关的RDF数据，这些数据形成了对给定资源的详细描述。
 - ▶ **CONSTRUCT**: 则根据查询图的结果生成RDF。

数据查询：SELECT语句

- ▶ SELECT语句的基本语法为：

SELECT 变量1 变量2 ...

WHERE 图模式

[修饰符]

- ▶ SELECT子句中的“变量1 变量2 ...”和SQL中的“列1, 列2, ...”的含义类似，表示要查询的目标。不同之处在于，SQL处理的数据存储于结构化的二维表中，语句中的“列1, 列2, ...”和数据表中的列完全对应，因此查询的目标具有明确的语义；而SPARQL处理的数据则具有更加灵活的存储结构，“变量1 变量2 ...”在知识图谱中没有直接的对应。

数据查询：SELECT语句

- ▶ SELECT语句的基本语法为：

SELECT 变量1 变量2 ...

WHERE 图模式

[修饰符]

- ▶ WHERE子句用于为SELECT子句中的变量提供约束，查询结果必须完全匹配该子句给出的图模式。图模式主要由两类元素组成：
 - ▶ 一类是三元组，例如 “?x a Person” 表示变量 ?x 必须是Person的一个实例；
 - ▶ 另一类是通过FILTER关键字给出的条件限制，这些条件包括数字大小的限制、字符串格式的限制等。
- ▶ 修饰符是可选项，用于对查询结果做一些特殊处理，例如常见的修饰符有用于表示排序的ORDER子句、用于限制结果数量的Limit子句等。

数据查询：SELECT语句

- 例：查询图中创作时间是“1959”的节点。其对应的SPARQL语句为：

prefix ns: <http://example.org/ns#>

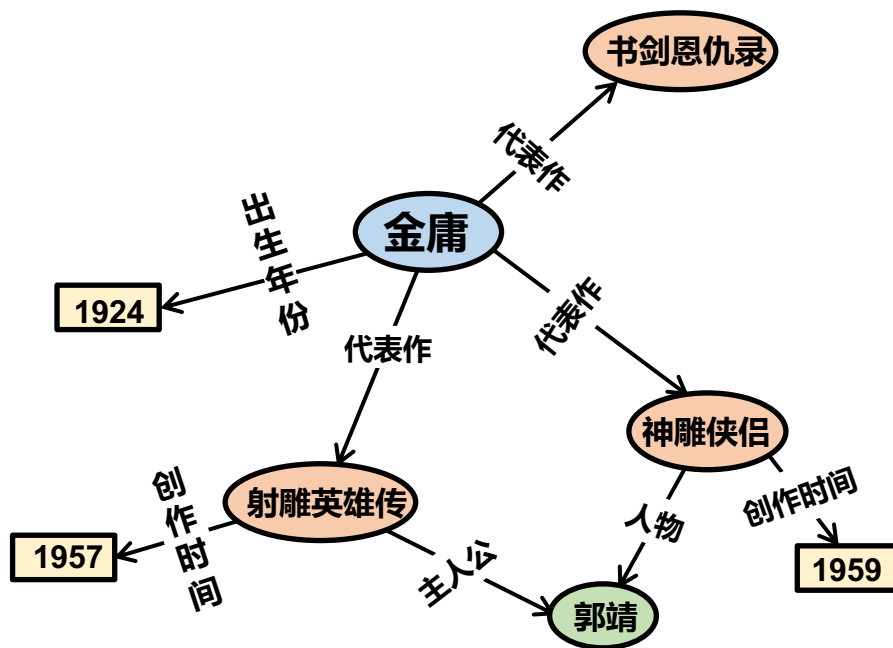
SELECT ?s {

?s ns: 创作时间 ns: 1959 .

}

- 查询结果为：

S
神雕侠侣



数据查询：ASK语句

- ▶ ASK语句用于测试知识图谱中是否存在满足给定图模式的数据，其基本语法为：

ASK 图模式

- ▶ 例：查询图中是否存在创作时间为“1957”的节点，其对应的SPARQL语句为：

```
prefix ns: <http://example.org/ns#>
```

```
ASK { ?s ns: 创作时间 ns: 1957 . }
```

- ▶ 该语句返回的结果为 “yes”；如果数据中没有对应的节点，查询的结果将会返回 “no”。

数据查询：DESCRIBE语句

- ▶ DESCRIBE语句用于获取和给定资源相关的数据，其基本语法为：

DESCRIBE 资源或变量

[**WHERE** 图模式]

- ▶ DESCRIBE后既可以跟确定的资源标识符，也可以跟变量；
- ▶ WHERE子句是可选的，用于限定变量需要满足的图模式。
- ▶ 获取出生年份为 "1924" 的节点的所有信息，其对应的SPARQL语句为：

prefix ns: <http://example.org/ns#>

DESCRIBE ?s **WHERE** { ?s ns: 出生年份 ns: 1924 . }

- ▶ 执行结果为：

@prefix ns: <http://example.org/ns#> .

ns: 金庸 ns: 代表作 ns: 书剑恩仇录

ns: 金庸 ns: 代表作 ns: 神雕侠侣

ns: 金庸 ns: 出生年份 ns: 1924

ns: 金庸 ns: 代表作 ns: 射雕英雄传

数据查询：CONSTRUCT语句

- ▶ CONSTRUCT语句用于生成满足图模式的RDF图，其基本语法为：

CONSTRUCT RDF图模板

WHERE 图模式

- ▶ RDF图模板确定了生成的RDF图所包含的三元组类型，它由一组三元组构成，每个三元组既可以是包含变量的三元组模板，也可以是不包含变量的事实三元组；图模式则和上述SELECT等语句中介绍的相同，用于约束语句中的变量。
- ▶ 该语句产生结果的基本流程为：首先执行WHERE子句，从知识图谱中获取所有满足图模式的变量取值；然后针对每一个变量取值，替换RDF图模板中的变量，生成一组三元组。

数据查询：CONSTRUCT语句

- 例：在图上执行如下SPARQL语句：

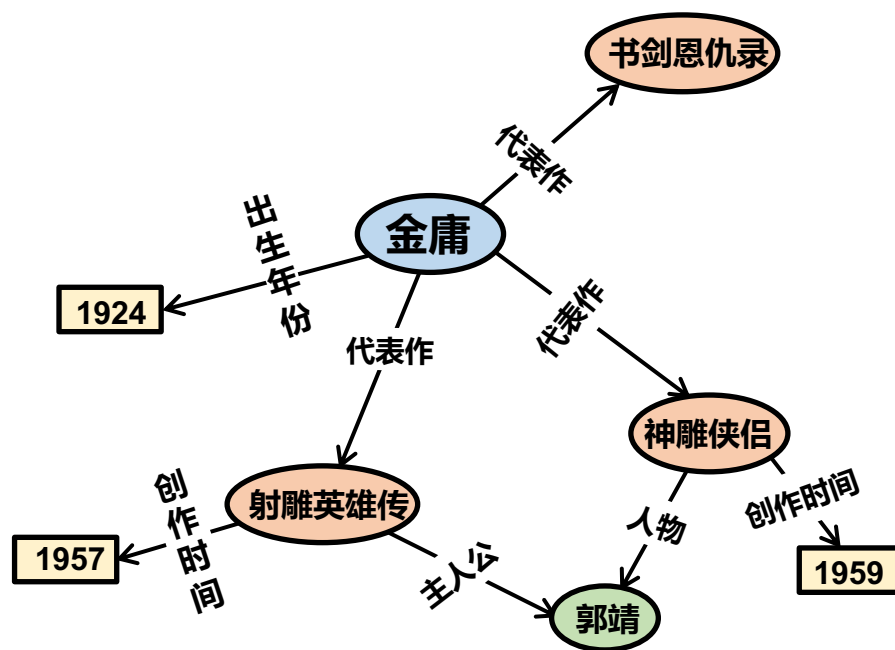
```
prefix ns: <http://example.org/ns#>
```

```
CONSTRUCT {  
  ?s ns: 本名 ns:查良镛 .  
}
```

```
WHERE {  
  ?s ns: 出生年份 ns: 1924 .  
}
```

- 执行结果为：

```
@prefix ns: <http://example.org/ns#> .  
ns: 金庸 ns:本名 ns:查良镛 .
```

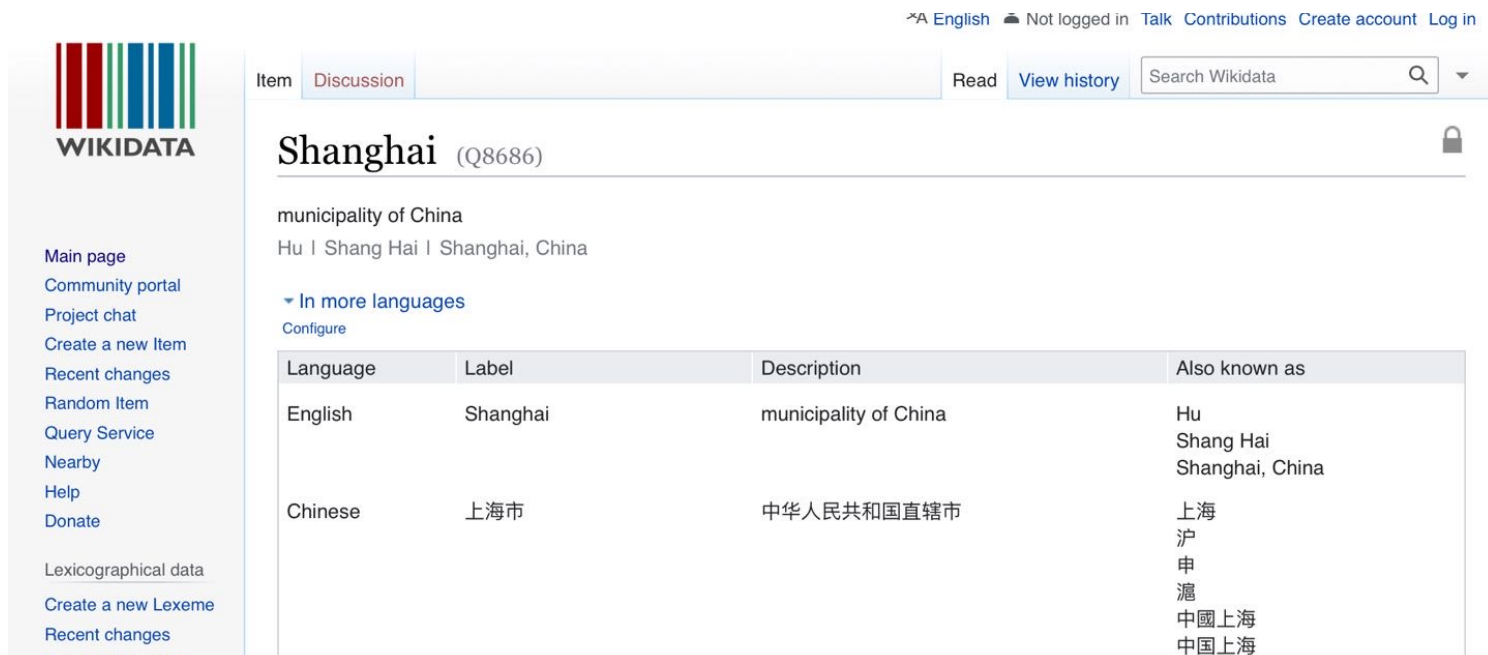


维基数据查询实例

- ▶ 下面通过维基数据查询 “上海市人口最多的是哪一个地区”， 进一步学习 SPARQL 语法。
- ▶ 首先，进入维基数据网站 (<https://www.wikidata.org>)，在页面顶部的搜索栏，搜索 “上海”。

The screenshot shows the Wikidata website interface. At the top, there is a navigation bar with links for English, Not logged in, Talk, Contributions, Create account, and Log in. Below this is a search bar containing the text '上海'. To the left of the search bar are tabs for Main Page, Discussion, Read, View source, and View history. The search results are displayed on the right side of the page, showing a list of items related to Shanghai. The items include: '上海高等教育系统教授录' (book published in 1988), '上海浦东国际机场鼠形动物及体外寄生...' (scientific article), 'Shanghai (上海)' (municipality of China), 'Shanghai Jiao Tong University (上海...)' (university in Shanghai, China), 'Shanghai Metro (上海地鐵)' (metro system in Shanghai, China), and 'Shanghai Stock Exchange (上海證券...)' (stock exchange that is based in the city ...). The 'Welcome to Wikidata' banner is visible in the background, stating 'the free knowledge base with 96,956,536 data items that anyone can edit'. The banner also includes links for Introduction, Project Chat, Community Portal, and Help.

- ▶ 然后进入上海市的页面。



Wikidata interface showing the page for Shanghai (Q8686). The page includes a sidebar with navigation links, a top navigation bar, and a main content area with a table of multilingual labels and descriptions.

Language	Label	Description	Also known as
English	Shanghai	municipality of China	Hu Shang Hai Shanghai, China
Chinese	上海市	中华人民共和国直辖市	上海 沪 申 滬 中國上海 中国上海

- ▶ 这时，留意一下这个页面的URL：https://www.wikidata.org/wiki/Q8686
- ▶ 上面URL最后结尾的Q8686，就是上海市这个条目在维基数据的编号（即主语）。

维基数据查询实例

- ▶ 接着，页面向下滚动，找到“contains administrative territorial entity”（所包含的行政实体）这个部分，它列出了上海市下辖的各个地区。

contains administrative territorial entity	Huangpu District
	▶ 1 reference
	Xuhui District
	▶ 1 reference
Changning District	▶ 1 reference
	Jing'an District
	▶ 1 reference

- ▶ 点击“contains administrative territorial entity”这个标题，进入它的页面，也留意一下 URL，<https://www.wikidata.org/wiki/Property:P150>
- ▶ 上面URL最后结尾的P150，就是“所包含的行政实体”这个谓语动词的编号

维基数据查询实例

- ▶ 现在，就可以开始查询了。进入维基数据的在线查询页面
query.wikidata.org



维基数据查询实例

















- ▶ 在查询框里面，输入下面的 SPARQL 语句。

```
SELECT ?area
WHERE {
    wd:Q8686 wdt:P150 ?area .
}
```

- ▶ 上面代码要求返回变量 ?area，该变量必须满足主语“上海市”(wd:Q8686)和谓语“所包含的行政实体”(wdt:P150)。前缀wd表示这是维基数据的条目，而前缀wdt表示这是维基数据定义的谓语关系。

维基数据查询实例

- ▶ 点击左侧边栏的三角形运行按钮，就可以在页面下方得到查询的结果。

area
 wd:Q57002
 wd:Q125378
 wd:Q660185
 wd:Q660626
 wd:Q660789
 wd:Q660952
 wd:Q661364
 wd:Q661533
 wd:Q661695
 wd:Q661828
 wd:Q662101
 wd:Q662241
 wd:Q662380
 wd:Q662548
 wd:Q662694
 wd:Q788812

- ▶ 从前图可以看到，返回的都是条目的编号。修改一下查询语句，增加一栏文字标签。

```
SELECT ?area ?areaLabel
WHERE {
    wd:Q8686 wdt:P150 ?area .
    ?area rdfs:label ?areaLabel .
    FILTER(LANGMATCHES(LANG(?areaLabel), "zh-CN"))
}
```

- ▶ 上面代码中，增加了一个返回的变量 ?areaLabel，该变量是前一个变量 ?area 的文字标签（满足谓语 rdfs:label），同时增加了一个过滤语句FILTER，要求只返回中文标签。

- ▶ 运行这段查询，就可以看到每个地区的中文名字了。

area	areaLabel
 wd:Q57002	徐汇区
 wd:Q125378	浦东新区
 wd:Q660185	黄浦区
 wd:Q660626	长宁区
 wd:Q660789	静安区
 wd:Q660952	普陀区
 wd:Q661364	虹口区
 wd:Q661828	宝山区

- ▶ 接着，再增加一个人口变量 ?popTotal，返回每个地区的人口总数。

```
SELECT ?area ?areaLabel ?popTotal
WHERE {
    wd:Q8686 wdt:P150 ?area .
    ?area rdfs:label ?areaLabel .
    FILTER(LANGMATCHES(LANG(?areaLabel), "zh-CN"))

    ?area wdt:P1082 ?popTotal .
}
```


- ▶ 运行这段代码，就可以看到人口总数了。

area	areaLabel	popTotal
 wd:Q57002	徐汇区	1085130
 wd:Q125378	浦东新区	5681512
 wd:Q660185	黄浦区	678670
 wd:Q660626	长宁区	690571
 wd:Q660789	静安区	1077284
 wd:Q660952	普陀区	1288881
 wd:Q661364	虹口区	852476
 wd:Q661533	杨浦区	1313222

- ▶ 然后，增加一个排序子句order by，按照人口的倒序排序。

```
SELECT ?area ?areaLabel ?popTotal
WHERE {
    wd:Q8686 wdt:P150 ?area .
    ?area rdfs:label ?areaLabel .
    FILTER(LANGMATCHES(LANG(?areaLabel), "zh-CN"))

    ?area wdt:P1082 ?popTotal .
}
ORDER BY desc(?popTotal)
```

► 运行结果如下：

area	areaLabel	popTotal
 wd:Q125378	浦东新区	5681512
 wd:Q661695	闵行区	2508000
 wd:Q661828	宝山区	2030500
 wd:Q662380	松江区	1582398
 wd:Q661533	杨浦区	1313222
 wd:Q660952	普陀区	1288881
 wd:Q662548	青浦区	1215000
 wd:Q57002	徐汇区	1085130

- ▶ 最后，加上一个limit 1子句，只返回第一条数据。

```
SELECT ?area ?areaLabel ?popTotal
WHERE {
    wd:Q8686 wdt:P150 ?area .
    ?area rdfs:label ?areaLabel .
    FILTER(LANGMATCHES(LANG(?areaLabel), "zh-CN"))

    ?area wdt:P1082 ?popTotal .
}
ORDER BY desc(?popTotal)
limit 1
```

- ▶ 这样就得到了上海市人口最多的地区。

area	areaLabel	popTotal
 wd:Q125378	浦东新区	5681512

- ▶ 查询金庸的代表作