数据集成作业三：可视化

0 小组成员

191250014 陈子鸿

191250109 潘昱光

1 数据样貌

数据来源是院系A、院系B、院系C三个院系教学管理系统的数据库系统。

其中：

- A 的 数据库系统使用 SqlServer
- B 的数据库系统使用 Oracle
- C 的数据库系统使用 MySQL

每个数据库系统包含学生、课程和选课信息。学生可以选择除了自己院系之外的课程。

根据作业二的要求，先有一个数据集成系统，使用xml技术汇总了院系A、院系B、院系C的数据，并统一了格式。

2 可视化需求

1. 能够展示学生的不同院系来源
2. 能够展示选课关系，即学生X选择了课程Y
3. 能够实现基本的查询、分类功能，如选择课程Y的学生有哪些，分别来自哪些院系

3 可视化实现思路

3.1 使用知识图谱网络可视化

本次作业决定使用知识图谱进行可视化，用节点和关系展示学生和课程之间的选择关系。

3.1.1 使用知识图谱可视化的优点

1. 知识图谱网络可视化可以帮助管理员更直观地理解学生和课程之间的选择关系。
2. 通过知识图谱网络可视化，管理员可以更方便地对选课情况进行分析。

3. 知识图谱网络可视化为管理员提供了一种直接交互的方式，可以帮助管理员快速定位和查找需要的信息。

3.2 定义可视化的实体类型和关系

实体类型定义如下：

- 学生
  - 学号
  - 姓名
  - 性别
- 账户
  - 账户名
  - 描述
  - 密码
  - 权限
- 院系
  - 院系名
- 课程
  - 学分
  - 课时
  - 课程名称
  - 授课老师
  - 课程编号
  - 授课地点

关系定义如下：

实体	关系	实体
学生	属于	院系
学生	拥有	账户
学生	选择	课程
院系	开设	课程

其中，有四类实体，分别是学生、账户、院系和课程。学生有属性学号、姓名和性别，账户有属性账户名、描述、密码、权限，院系有属性院系名，课程有属性课时、课程名称、授课老师、课程编号、授课地点。

有四类关系：学生属于院系、学生拥有账户、学生选择课程、院系开设课程。

使用OWL知识图谱本体描述语言描述如下：

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" ><!ENTITY xsd "http://www.w3.o
]>
<rdf:RDF xmlns="http://example.org/ontology#"xml:base="http://example.org/ontolo

<!-- 学生 -->
<owl:Class rdf:about="#Student"/>

<owl:ObjectProperty rdf:about="#hasAccount">
    <rdfs:domain rdf:resource="#Student"/>
    <rdfs:range rdf:resource="#Account"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#selectsCourse">
    <rdfs:domain rdf:resource="#Student"/>
    <rdfs:range rdf:resource="#Course"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:about="#studentID">
    <rdfs:domain rdf:resource="#Student"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#studentName">
    <rdfs:domain rdf:resource="#Student"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#studentGender">
    <rdfs:domain rdf:resource="#Student"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<!-- 账户 -->
<owl:Class rdf:about="#Account"/>

<owl:DatatypeProperty rdf:about="#accountName">
    <rdfs:domain rdf:resource="#Account"/>
```

```
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#accountDescription">
    <rdfs:domain rdf:resource="#Account"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#accountPassword">
    <rdfs:domain rdf:resource="#Account"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#accountPermission">
    <rdfs:domain rdf:resource="#Account"/>
    <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>

<!-- 院系 -->
<owl:Class rdf:about="#Department"/>

<owl:DatatypeProperty rdf:about="#departmentName">
    <rdfs:domain rdf:resource="#Department"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<!-- 课程 -->
<!-- …… -->
```

3.3 使用图数据库Neo4J同步数据

Neo4j是知识图谱常用的数据库，是一种高性能的图形数据库，它使用图形模型来表示和存储数据。在传统的关系型数据库中，数据通常以表格的形式组织，而在图形数据库中，数据以节点和边缘的形式组织，这些节点和边缘可以非常自然地表示真实世界中的关系。

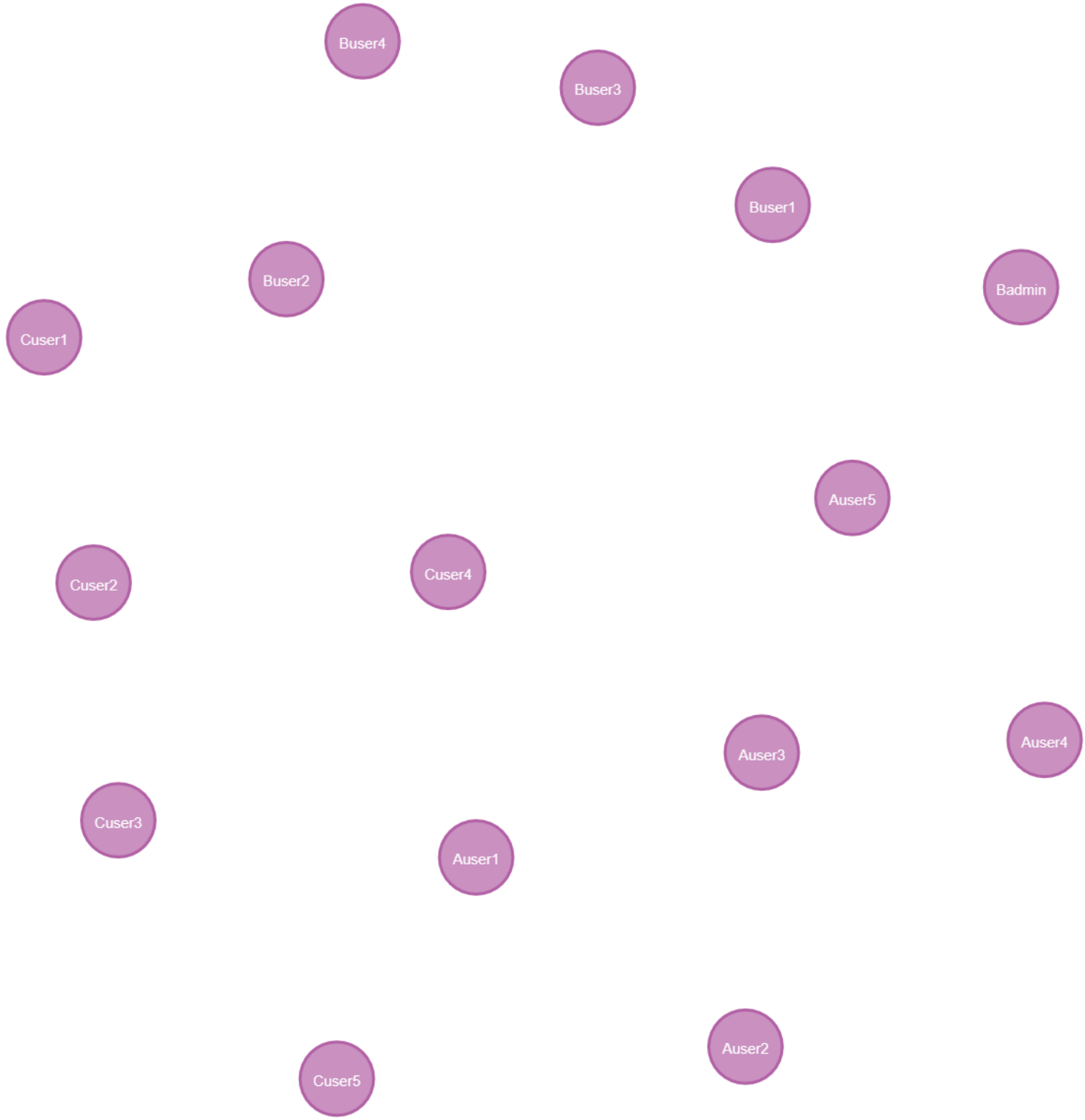
我们使用Cypher语言将集成系统中的学生选课数据导入Neo4j中。

- 导入账户实体

```
// 账户
MERGE (a1:Account {账户名: 'Auser1', 描述: "院系A账户",  密码: '123456', 权限: '
MERGE (a2:Account {账户名: 'Auser2', 描述: "院系A账户",  密码: '123456', 权限: '
MERGE (a3:Account {账户名: 'Auser3', 描述: "院系A账户",  密码: '123456', 权限: '
MERGE (a4:Account {账户名: 'Auser4', 描述: "院系A账户",  密码: '123456', 权限: '
```

```
MERGE (a5:Account {账户名: 'Auser5', 描述: "院系A账户",  密码: '123456', 权限: '
MERGE (b1:Account {账户名: 'Badmin', 描述: "院系B账户", 密码: '123456', 级别: 1,
MERGE (b2:Account {账户名: 'Buser1', 描述: "院系B账户", 密码: '123456', 级别: 2,
MERGE (b3:Account {账户名: 'Buser3', 描述: "院系B账户", 密码: '123456', 级别: 2,
MERGE (b4:Account {账户名: 'Buser4', 描述: "院系B账户", 密码: '123456', 级别: 2,
MERGE (b5:Account {账户名: 'Buser2', 描述: "院系B账户", 密码: '123456', 级别: 2,
MERGE (c1:Account {账户名: 'Cuser1', 描述: "院系c账户", passwd: '123456', Creat
MERGE (c2:Account {账户名: 'Cuser2', 描述: "院系c账户", passwd: '234567', Creat
MERGE (c3:Account {账户名: 'Cuser3', 描述: "院系c账户", passwd: '345678', Creat
MERGE (c4:Account {账户名: 'Cuser4', 描述: "院系c账户", passwd: '456789', Creat
MERGE (c5:Account {账户名: 'Cuser5', 描述: "院系c账户", passwd: '567890', Creat
```

如图，我们添加了15个账户节点：



账户的节点信息如下：

Node properties

Account

<id>	0
密码	123456
描述	院系A账户
权限	admi
账户名	Auser1

Node properties

Account

<id>	8
客体	20220004
密码	123456
描述	院系B账户
级别	2
账户名	Buser4

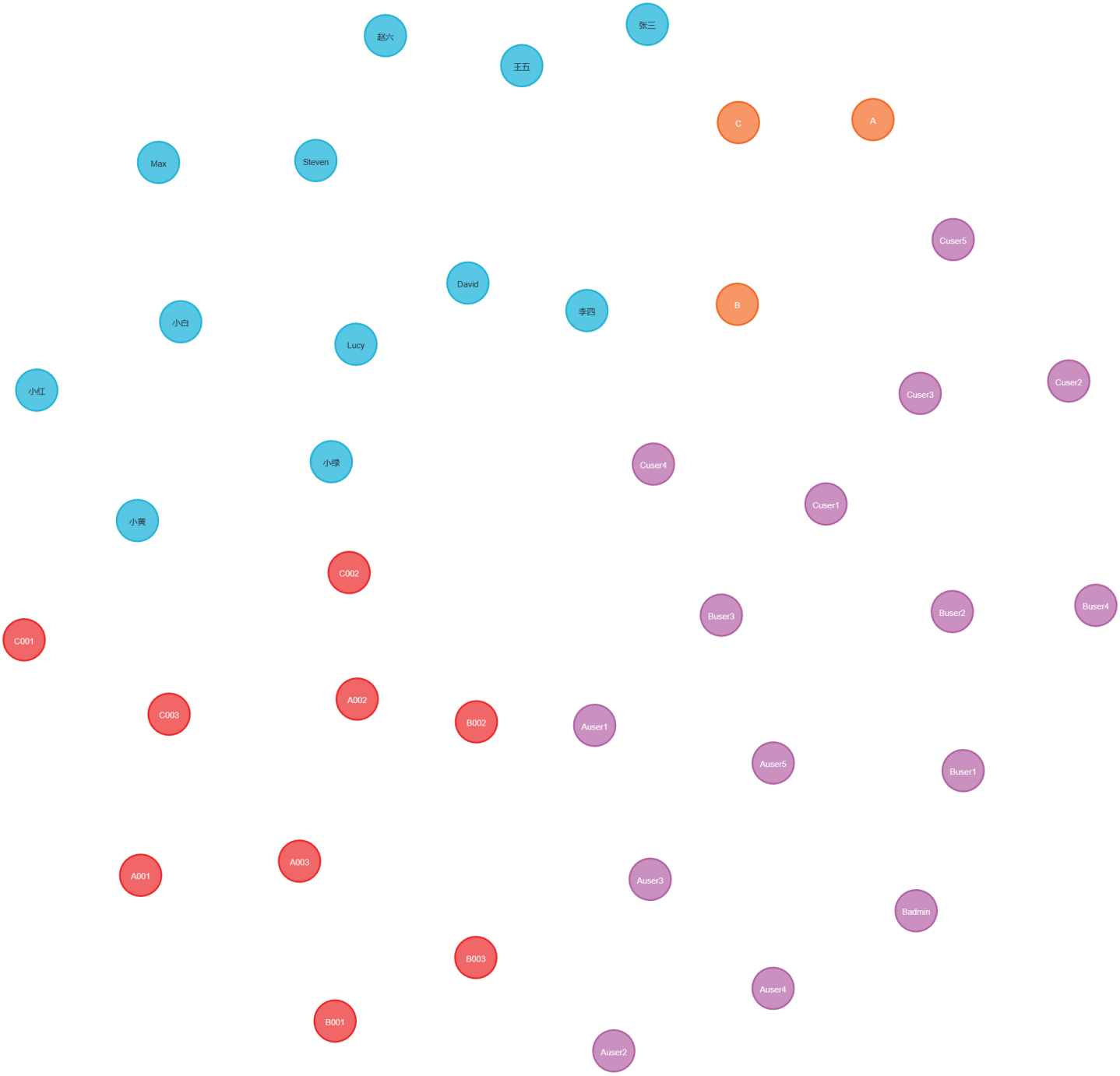
Node properties

Account

<id>	11
CreateDate	"2022-11-12"
passwd	234567
描述	院系c账户
账户名	Cuser2

- 导入院系、学生和课程

```
MERGE (A:Department{院系名:'A'});
MERGE (B:Department{院系名:'B'});
MERGE (C:Department{院系名:'C'});

MERGE (AStu1: Student{学号:'20210001', 姓名:'张三', 性别:'男'}); // account user3
MERGE (AStu2: Student{学号:'20210002', 姓名:'李四', 性别:'男'}); // account user4
MERGE (AStu3: Student{学号:'20210003', 姓名:'王五', 性别:'女'}); //account user5
MERGE (AStu4: Student{学号:'20210004', 姓名:'赵六', 性别:'男'}); //account user2

MERGE (BStu1: Student{学号:'20220001', 姓名:'David', 性别:'男'}); // account user1
MERGE (BStu2: Student{学号:'20220002', 姓名:'Steven', 性别:'男'}); // account user2
MERGE (BStu3: Student{学号:'20220003', 姓名:'Max', 性别:'男'}); //account user3
MERGE (BStu4: Student{学号:'20220004', 姓名:'Lucy', 性别:'女'}); //account user4

MERGE (CStu1: Student{学号:'20230001', 姓名:'小白', 性别:'男'}); // account user1
MERGE (CStu2: Student{学号:'20230002', 姓名:'小红', 性别:'男'}); // account user2
MERGE (CStu3: Student{学号:'20230003', 姓名:'小绿', 性别:'女'}); //account user3
MERGE (CStu4: Student{学号:'20230004', 姓名:'小黄', 性别:'男'}); //account user4

MERGE (ACour1: Course{课程编号: 'C001', 课程名称:'计组', 学分:'3', 授课老师: '任老师'
MERGE (ACour2: Course{课程编号: 'C002', 课程名称:'数据结构', 学分:'4', 授课老师: '刘
MERGE (ACour3: Course{课程编号: 'C003', 课程名称:'数据库', 学分:'3', 授课老师: '刘老

MERGE (BCour1: Course{课程编号: 'A001', 课程名称:'中国艺术史', 学分:'3', 授课老师: '
MERGE (BCour2: Course{课程编号: 'A002', 课程名称:'西方艺术史', 学分:'3', 授课老师: '
MERGE (BCour3: Course{课程编号: 'A003', 课程名称:'美国艺术史', 学分:'3', 授课老师: '

MERGE (CCour1: Course{课程编号: 'B001', 课程名称:'宏观经济学', 学分:'4', 授课老师: '
MERGE (CCour2: Course{课程编号: 'B002', 课程名称:'微观经济学', 学分:'3', 授课老师: '
MERGE (CCour3: Course{课程编号: 'B003', 课程名称:'政治经济学', 学分:'2', 授课老师: '

```



- 连接关系

```
// 学生属于院系
MATCH (a:Student), (b:Department)
WHERE a.学号 = '20210001' AND b.院系名 = 'A'
CREATE (a)-[r:属于]->(b)
RETURN r

// 学生拥有账户
// ……

// 学生选择课程
// ……

```

```
13
14  // 院系开设课程
15  // ……
16
```

最终的可视化结果如下：

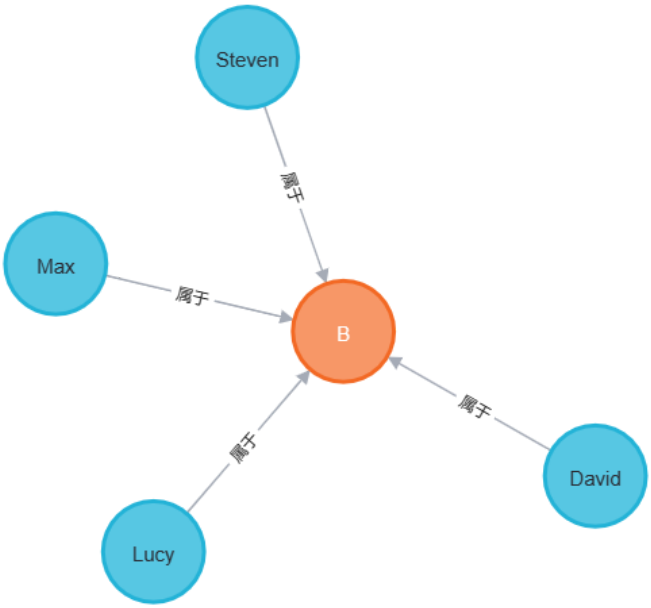
4 可视化筛选功能

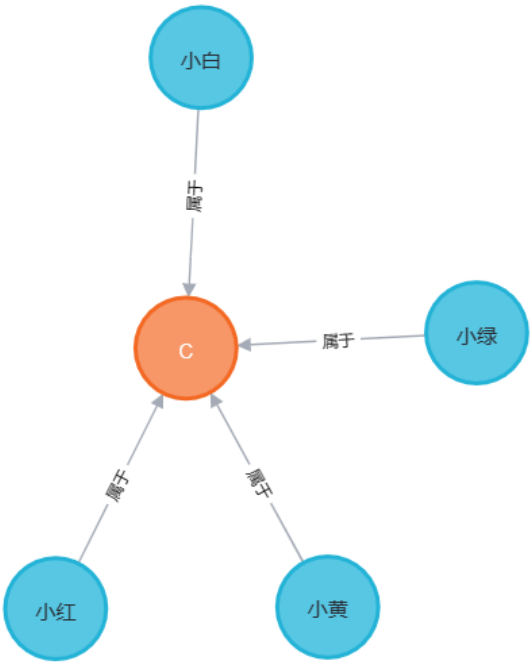
利用Neo4j的Cypher语句可以实现简单的查询可视化。

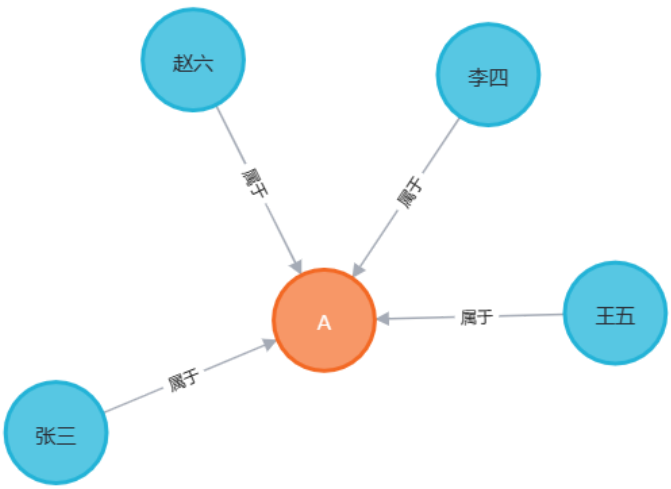
例如，如果想要得到A，B，C三个院系的学生的情况，可以用以下查询语句查询：

```
MATCH p=()-[r:`属于`]->() RETURN p LIMIT 25
```

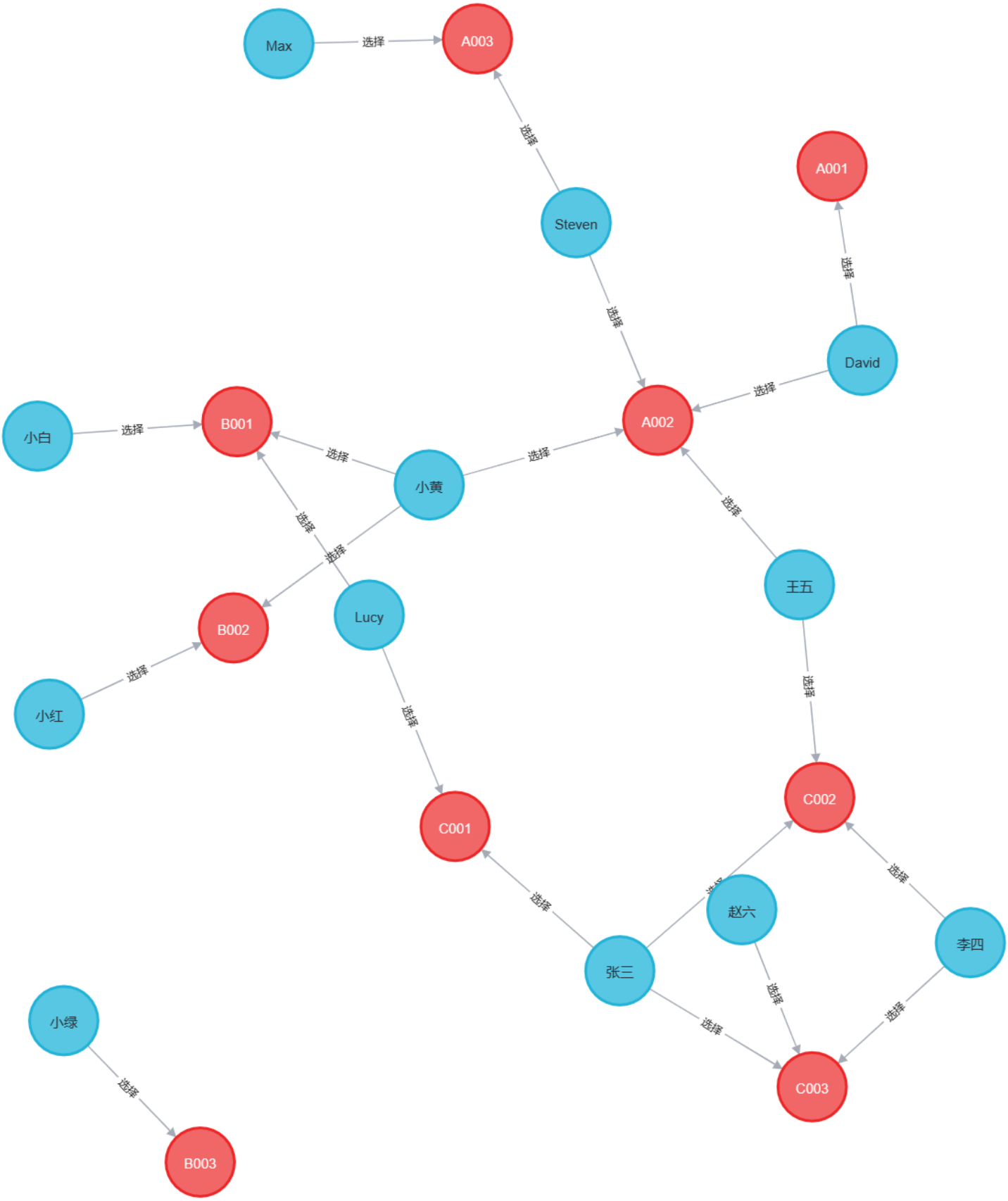
结果如下：


Steven
属于
Max
属于
B
属于
属于
David
Lucy

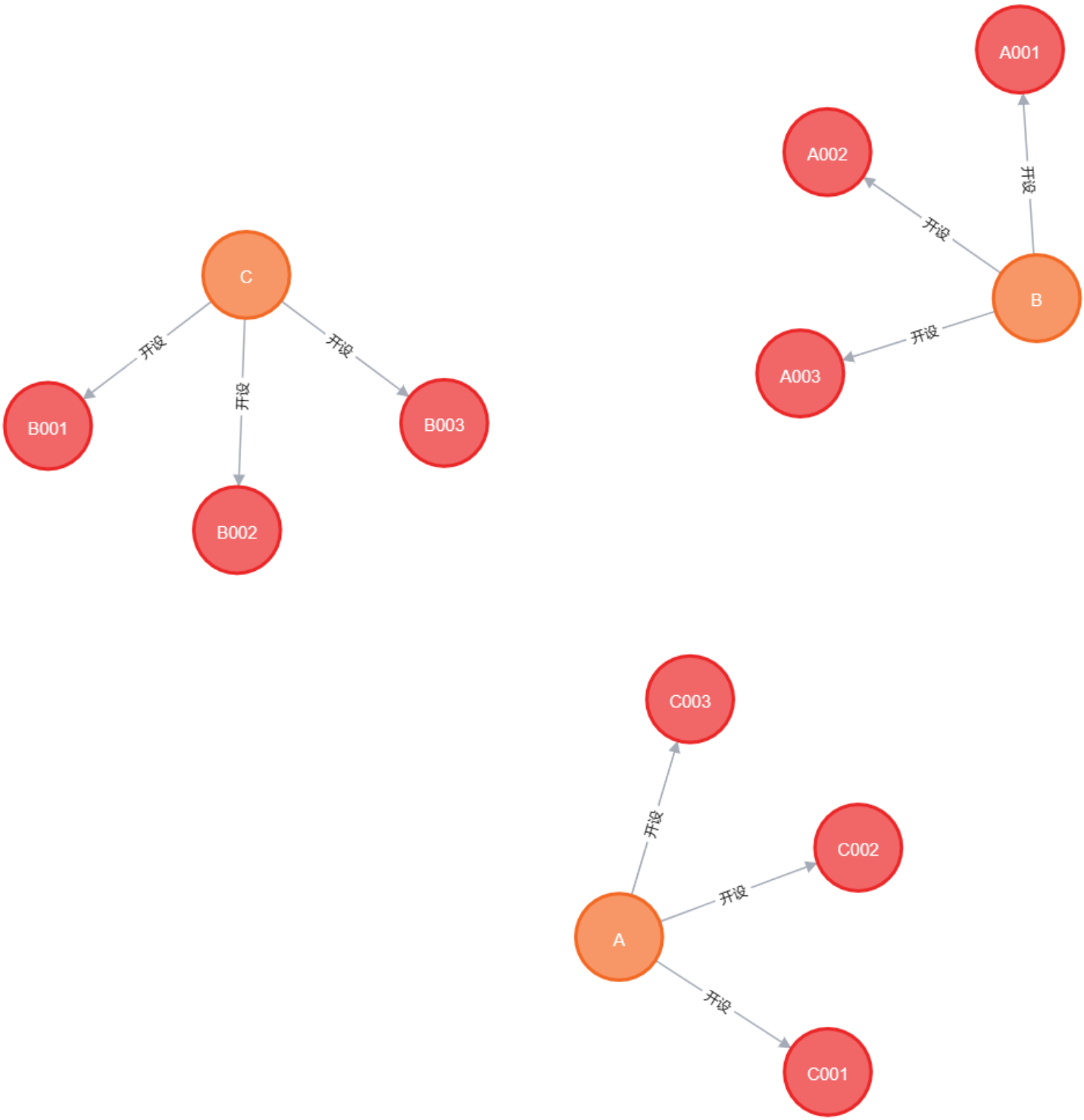

小白
属于
C
属于
小绿
属于
属于
小红
小黄


赵六
李四
属于
属于
A
属于
王五
属于
张三

同理，选课情况如下：


Max
选择
A003
选择
Steven
选择
A001
选择
David
选择
A002
小白
选择
B001
选择
小黄
选择
选择
王五
选择
B002
选择
Lucy
小红
选择
选择
C002
C001
选择
选择
赵六
选择
李四
张三
选择
选择
小绿
选择
C003
B003

院系开设课程情况如下：



还可以查询某个学生的课程选择情况：

```
1 MATCH (s:Student{姓名:"张三"})
2 OPTIONAL MATCH (s)-[r:选择]->(c:Course)
3 RETURN s.姓名, collect(c.课程名称) AS courses
```

结果如下：


neo4j$ MATCH (s:Student{姓名:"张三"}) OPTIONAL MATCH (s)-[r:选择]→(c:Course) RETURN s.姓名, collec…
Table
Text
Code
s.姓名
courses
1
"张三"
["数据库", "数据结构", "计组"]