# 图数据库 Neo4j 基础 演练

# 演练 1 Neo4j 介绍与使用

## 演练介绍

本次演练对图数据库 Neo4j 的概念与操作方法做基础介绍。

完成本次演练，可以对这Neo4j 有简单的基础了解，之后的演练也将在此基础上进行。

#### 技能点

* 图数据库的概念
* Neo4j 的安装
* Neo4j 数据迁移
* Neo4j Browser 的使用

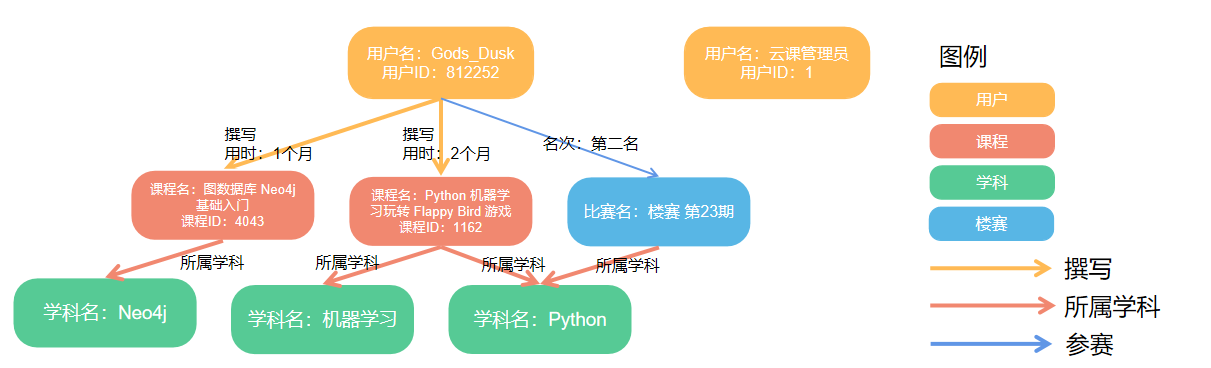
## 图数据库的概念

图数据库的基本概念是图，用一个简单的例子来介绍图：

#### 图例

如下就是一个图，图中包含的信息是：一位名为 Gods\_Dusk 的用户，发布了两门云课程，其中一门叫做 《Python 机器学习玩转 Flappy Bird 游戏》，所属学科是机器学习和 Python，另一门课程叫做《图数据库 Neo4j 基础》，所属学科为 Neo4j。同时该用户参加了第 23 期楼赛，获得了第二名。

另一名用户为云课管理员，用户 ID 为 1。



#### 节点

节点通常用于表示**实体**，实体的概念来自哲学，现实世界中的客观事物、抽象的事件均可称之为实体。节点是组成图的最小单位。

#### 标签

标签用于分类节点，如所有用来表示课程的节点都可以使用标签：**课程**进行标记。在 Neo4j 中，就可以根据不同标签类型进行特定节点的查询。

一个节点可以有多个标签。如用户 Gods\_Dusk，既是用户，也是课程发布者；用户 云课管理员，既是用户，也是公司员工。

#### 关系

图中的边表示关系，关系连接两个节点，如**撰写**关系，将《图数据库 Neo4j 基础》课程和用户 Gods\_Dusk 相连接起来。

关系具有方向性，如《图数据库 Neo4j 基础》属于 Neo4j 类型的课程，而反之不成立，但只需要注意到关系的方向，就不需要在相反的方向上添加重复的关系，即 Neo4j 类型的课程包含《图数据库 Neo4j 基础》。

被关系连接的两个节点，其中关系的起始节点被称为**出节点**，关系的终止节点被称为**入节点**。同时，一个节点的**出度**是指这个节点被多少关系作为出节点，同理，**入度**是指被多少个关系作为入节点。

特别的是，一个节点可以有指向自己的关系，比如用户 Gods\_Dusk **认识**自己。

#### 关系类型

关系与节点一样具有类型划分，但关系有且仅有一种关系类型。

#### 属性

属性是用于描述节点和关系特性的**名称 - 值**对。 如一个用户，具有属性：用户名 - Gods\_Dusk，用户 ID - 812252。

#### 路径与遍历

在图中，查询过程是路径搜索与遍历的过程，如 “Gods\_Dusk 发布过哪些课？”，为了获得答案，就需要先从图中找到用户名为 Gods\_Dusk 的节点，然后找到类型为撰写的关系，这些关系的出节点就是答案，在这里路径的搜索长度为 1。而问题“Gods\_Dusk 写过的课程都是属于哪些类型？”的搜索长度为 2。

## Neo4j 安装与使用

**默认环境（Windows）已经安装 Neo4j V5.x.x，可以直接使用，无需再次安装。**

安装过程（以Windows为例）：

前往 [官网](https://neo4j.com/download/) 下载对应操作系统的安装包（Zip）。

Windows需要下载压缩包，解压后进入 **bin** 目录启动程序。

### 启动 Neo4j

打开终端，输入

neo4j windows-service install

安装Windows Services。

然后，输入：

neo4j start

即可启动 Neo4j 服务。

同时输入：

neo4j status

可以查看 Neo4j 服务的启动情况。

打开浏览器，在地址栏输入 localhost:7474，就可以打开 Neo4j Browser（若网页提示连接失败，请等待片刻后再刷新网页）。Neo4j Browser 是 Neo4j 数据库面向开发人员的工具，允许开发人员执行查询语句并可视化结果。

首次进入 Neo4j Browser 时需要重置密码，系统默认密码为 neo4j，在本演练中，建议统一修改密码为 12345678。

## 数据迁移

Neo4j 提供了 **neo4j-admin** 来进行数据的导入导出，在迁移之前，需要停止 Neo4j 服务（neo4j stop）。

#### 数据导入

# 导出的 Neo4j 数据文件为：

neo4j.dump

# 停止 Neo4j 服务

sudo neo4j stop

# 导入到本地数据库中，默认数据库名为 neo4j，本处强制覆盖数据库的内容

注意只要路径（neo4j.dump 在 d:\neo4jdata 下），不要文件名

neo4j-admin.bat database load --overwrite-destination=true --from-path=d:/neo4jdata neo4j

# 因为当前 neo4j.dump 是旧版本的数据库，需要合并迁移到当前最新版本

neo4j-admin database migrate --force-btree-indexes-to-range --verbose neo4j

# 启动 Neo4j 服务

sudo neo4j start

#### 数据导出

# 停止 Neo4j 服务

sudo neo4j stop

# 导出数据到本地 d:/code 目录下

neo4j-admin database dump --verbose --overwrite-destination=true --to-path=d:/code neo4j

# 启动 Neo4j 服务

sudo neo4j start

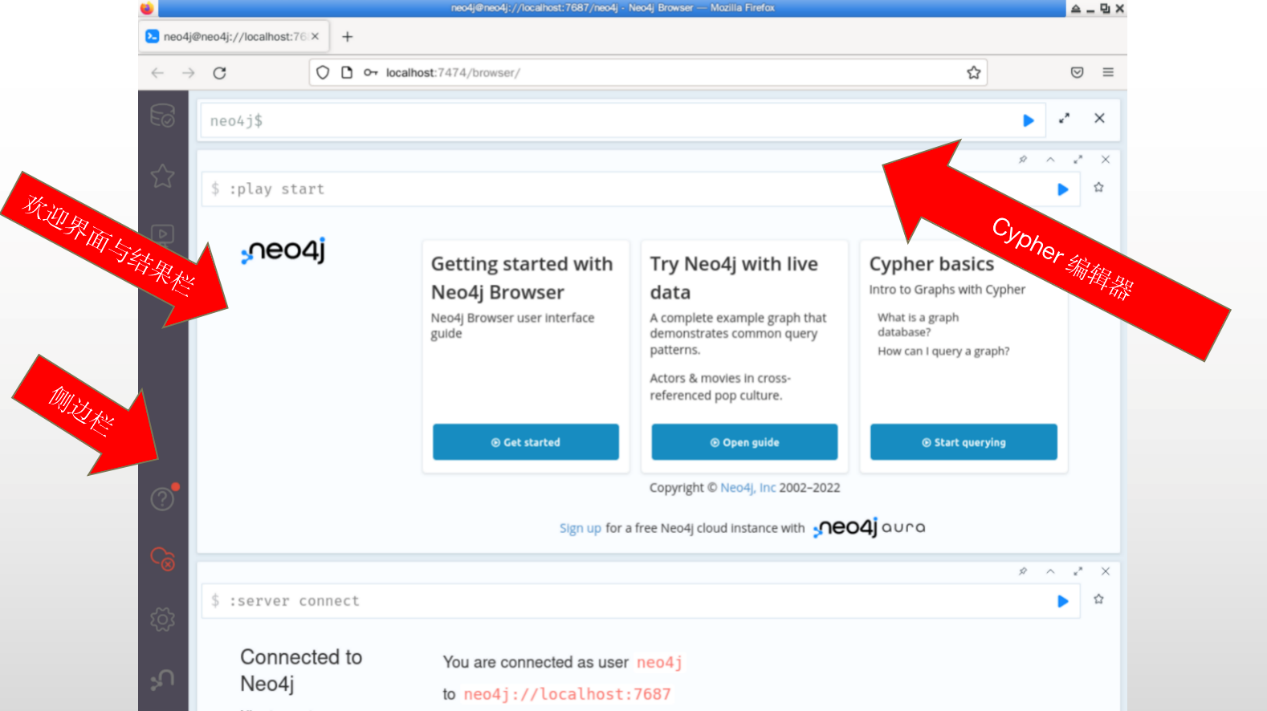
说明：上述命令中：

--verbose 用于显示命令执行过程中的详细信息，可以省略

neo4j 是 Neo4j 中的数据库名称

打开浏览器，输入地址 localhost:7474 进入 Neo4j Browser 重置密码，系统默认密码为 neo4j，密码为 12345678。

## Neo4j Browser

Neo4j Browser 中有 3 个模块，侧边栏，Cypher 编辑器与结果栏，在进入 Neo4j Browser 时结果栏会展示欢迎界面。 

### Cypher 编辑器

Cypher 是 Neo4j 数据库的查询语言， Cypher 编辑器则用于输入和运行 Cypher 语句。

在 Cypher 编辑器右侧有三个按钮。



其功能分别是：

* 运行输入框中的 Cypher 语句，若语句只有一行，则可以用回车键（enter）作为快捷键，若需要输入多行命令，使用 shift+enter 实现输入框内的换行，运行多行命令，需要单击运行按钮（或，使用 ctrl+enter快捷键）。
* 最大化 Cypher 编辑器。
* 清除输入框中的 Cypher 语句。

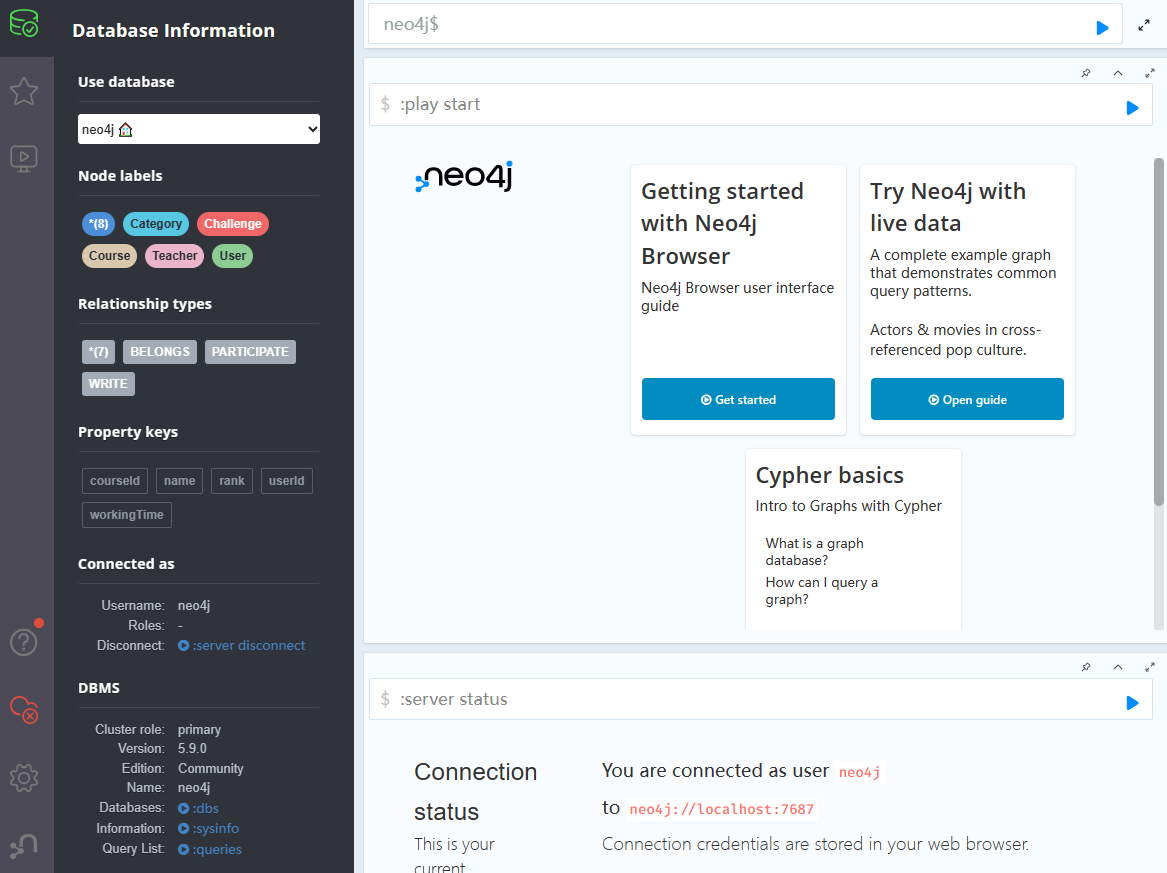
### 侧边栏

侧边栏包含一组按键，用于数据浏览和环境设置。从上至下分别是：

#### 数据库信息

数据库信息一栏提供了当前图数据库概览，并展示当前用户的角色和 Neo4j 版本信息。

前面，从其他数据库中导入了一些数据，所以在点击数据库信息展示按钮时，可以展示数据库中具体统计内容，如下图所示：

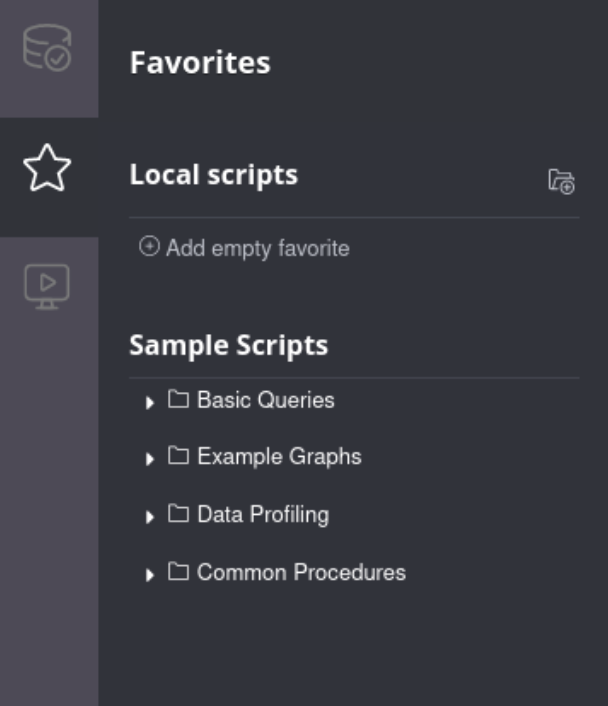


* **Node Labels**：展示数据库中节点数量与节点类型，在这里一共 8 个节点，节点类型有 **Course**，**User**，**Teacher**，**Challenge** 和 **Category**。
* **Relationship Types**：展示数据库中关系数量与关系类型，在这里一共 7 个关系，关系类型有 **WRITE**，**BELONGS**，**PARTICIPATE**。
* **Property Keys**：展示数据库中节点与关系的属性名，在这里属性名有 **name**，**courseId**，**workingTime**，**userId**，**rank**。
* **Connected as**：显示当前用户信息，若当前用户为管理员，则还可以进行成员管理操作。
* **DBMS**：显示 Neo4j 的版本与系统信息。

单击节点、关系、属性中显示的任何一个标签，Neo4j 会运行一个快捷查询，并在结果栏可视化查询结果。

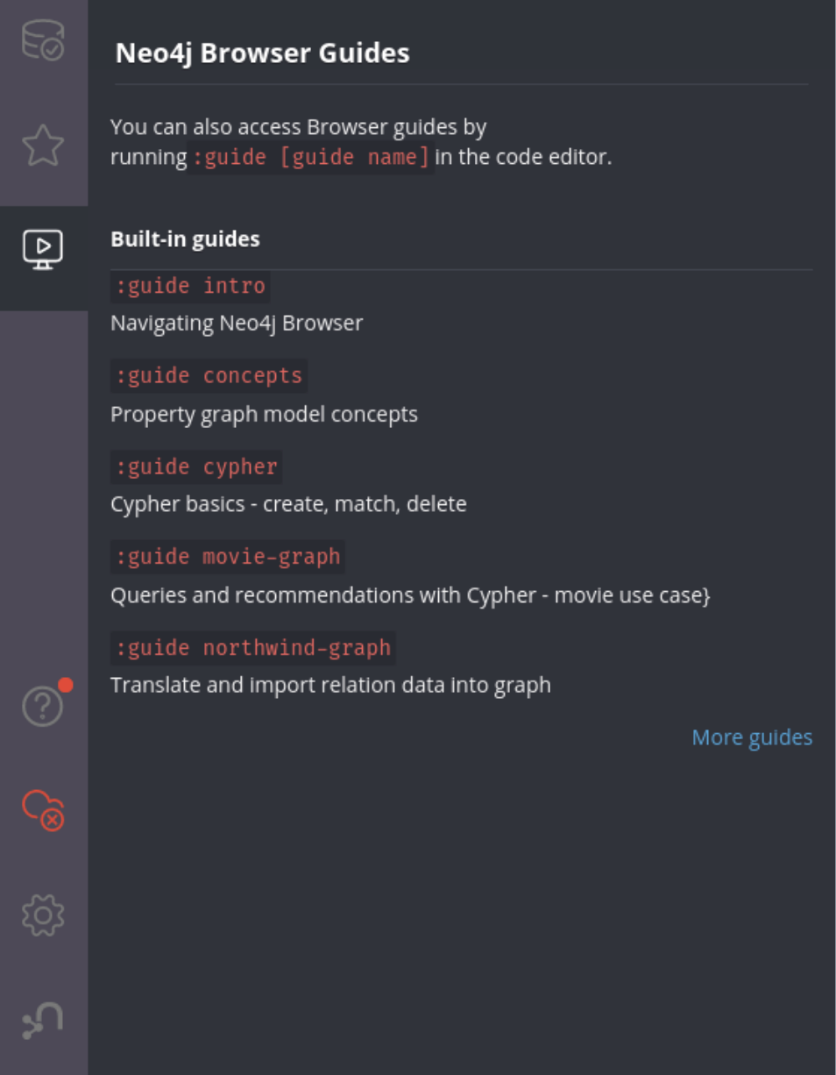
#### 收藏夹

收藏夹用于收藏 Cypher 命令，以便快捷使用。



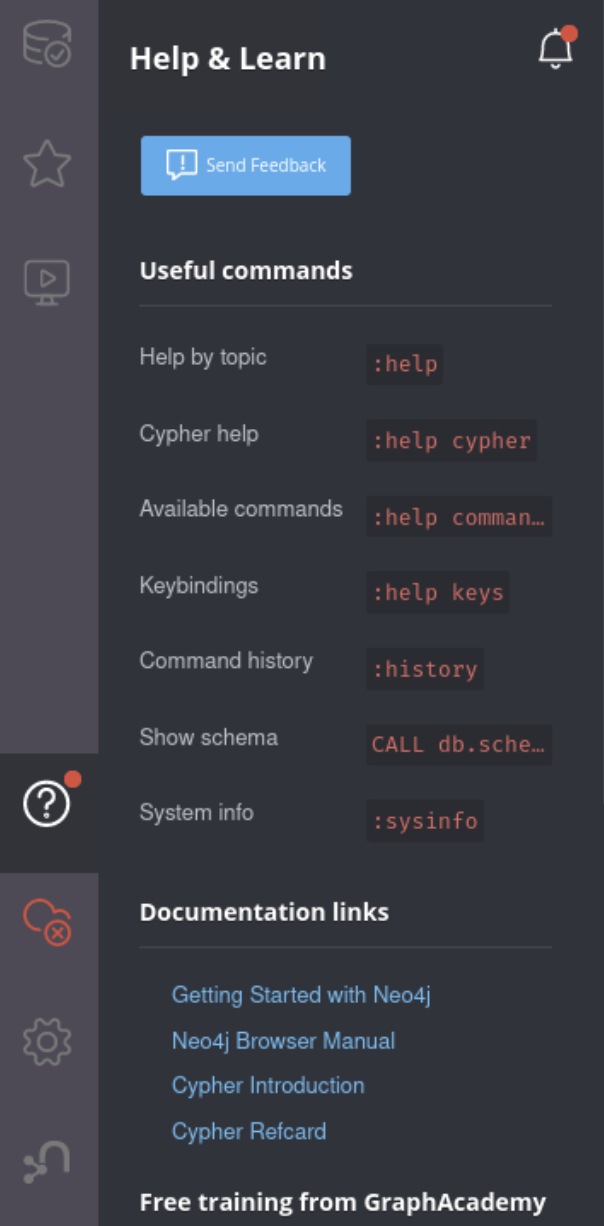
#### Neo4j Browser 指引

官方的 Neo4j Browser 使用方法指引。



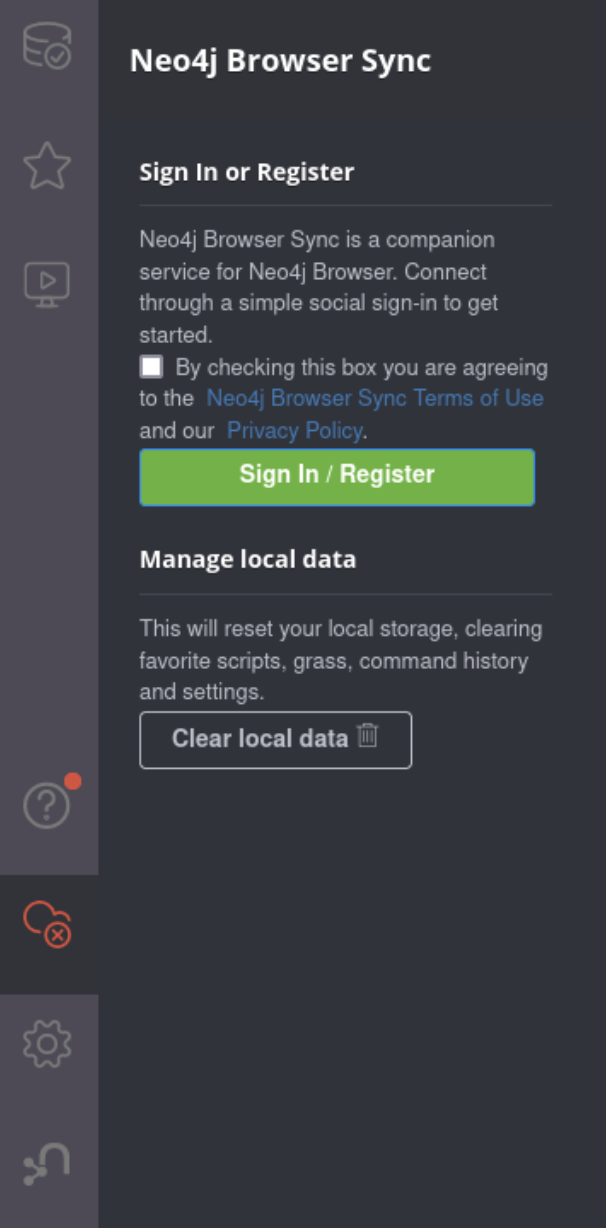
#### 文档与帮助

常见使用问题的解答，同时 Neo4j Browser 中直接内嵌了官方文档，点击相应标题就可以跳转到对应的官方文档。



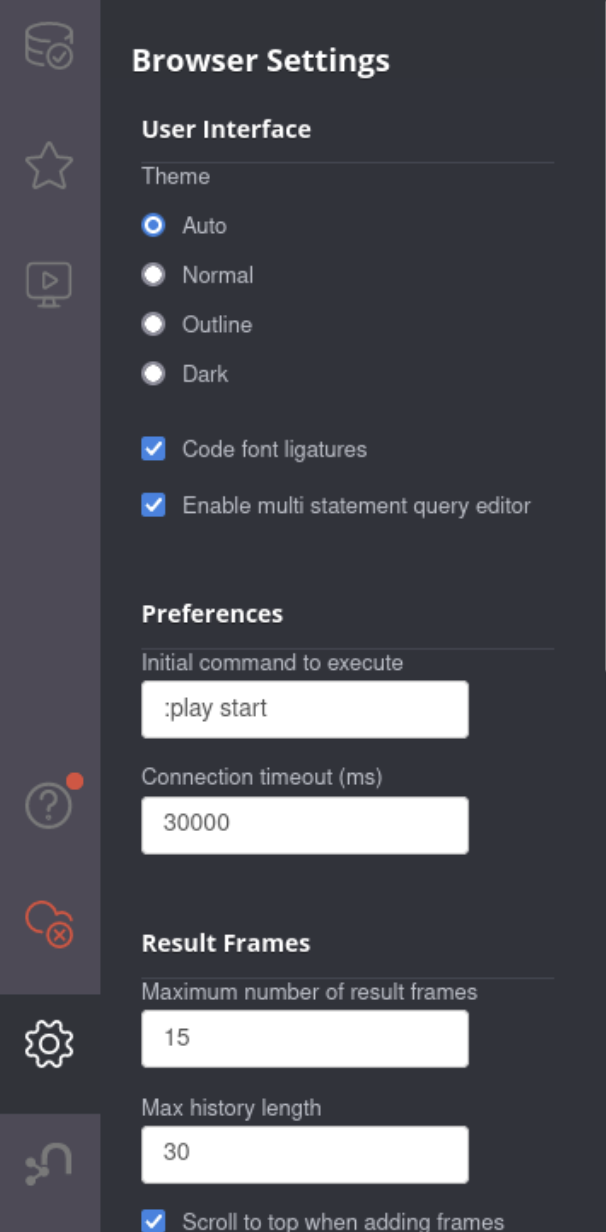
#### 数据库远程同步

清空本地数据库，与云端数据库同步。



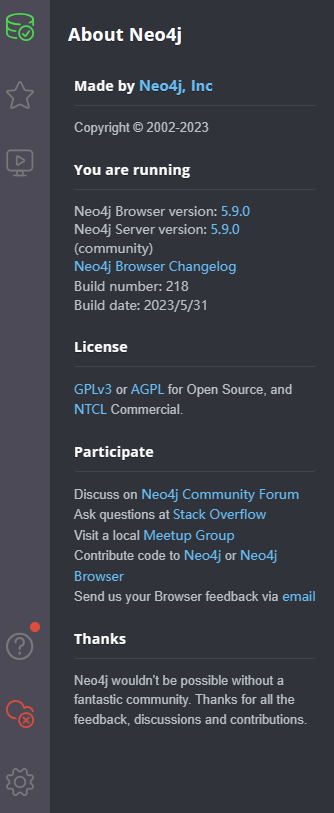
#### 设置

浏览器设置可对网页 UI，结果图形可视化进行相关的配置。



#### Neo4j 相关信息

显示关于 Neo4j 的相关信息，包括版本号、许可证、社区以及致谢。



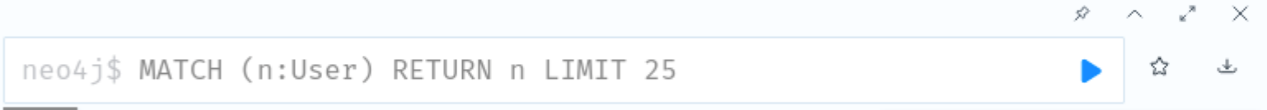
### 结果栏

Neo4j Browser 以结果帧的方式，在结果栏保存每一次运行结果，允许用户直接在结果帧中进行查询后的操作。

以点击侧边栏中节点 **User** 为例，在一个结果帧中，上方左侧记录了运行的 Cypher 语句 MATCH (n:User) RETURN n LIMIT 25，点击后会在 Cypher 编辑器中显示该语句，方便用户修改和复用。

右侧的按钮从上到下，从左到右分别是：

* 将结果帧钉在顶部
* 收起结果帧
* 全屏显示
* 关闭结果帧
* 重新运行
* 收藏 Cypher 语句
* 结果下载



结果帧下方的结果视图分别有：

#### 图视图

以图的形式显示结果，并允许配置节点、边、文字的显示风格（如颜色，大小，显示的属性），同时图视图允许用户对结果中的图进行拖动，隐藏和二次查找操作。视图的右边栏会显示对应节点和边的属性，其中 <id> 是 Neo4j 生成的自增主键。

图视图并不会始终出现在结果帧中，只有当查询结果可以表示为图时，才会有图视图。

#### 表视图

以 JSON 格式显示结果。



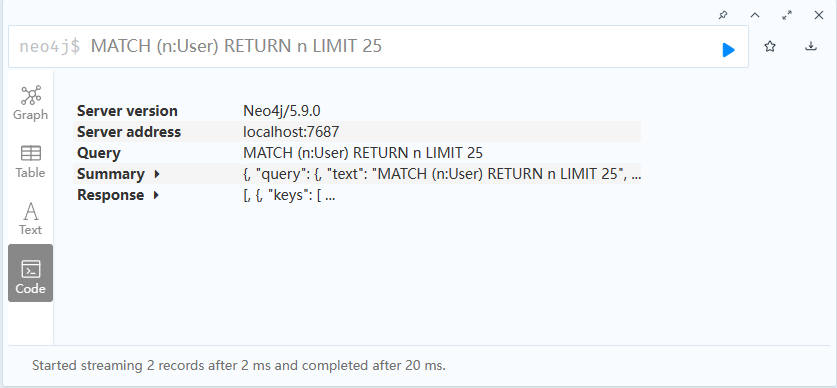
#### 文本视图

以纯文本表格的形式显示结果。



#### 代码视图

显示提交的请求，Neo4j 服务器版本和地址，以及响应结果。



### 演练小结

本演练是 Neo4j 的基础。在演练中，初步接触了图数据库，Neo4j 及 Cypher 的基本概念，实践了对 Neo4j 的 启动，停止和数据迁移，使用和查看 Neo4j Browser 等基本操作。

在下一演练，将在 Neo4j Browser 中使用 Cypher 编辑器进行 Cypher 语法语句的学习、演练。

# 演练 2 Cypher 图数据查询

## 演练介绍

本演练首先会介绍 Cypher 中图的表示方法，然后在 Neo4j Browser 中学习基本的图数据查询，同时引出 Cypher 中支持的数据类型与其相对应的运算操作。

在掌握上述技能点后，就可以演练 Cypher 的查询过滤方法与常用函数，最后将演练 Cypher 的复杂查询。

#### 技能点

* 图数据结构
* Cypher 数据类型与运算
* 图查询语法
* 常用函数
* 复杂查询

## 数据表示

上一个演练中介绍了图数据库中的包含的元素：节点，关系，标签，属性和路径，接下来将介绍在 Cypher 中如何表示这些元素。

#### 节点的表示

在 Cypher 中，使用圆括号 () 来表示节点，同时使用冒号 : 来隔开变量名与节点的标签或者节点的多个标签。多个节点之间用逗号 , 隔开。

() // 表示一个匿名节点

(n) // 表示一个变量名为 n 的节点

(n:User) // 表示一个变量名为 n，标签为 User 的节点

(:User) // 表示一个标签为 User 的匿名节点

(n:User:Teacher) // 表示一个变量名为 n，标签为 User 和 Teacher 的节点

(n:User), (m:Teacher) // 表示一个变量名为 n，标签为 User 的节点和一个变量名为 m，标签为 Teacher 的节点

#### 关系的表示

两个节点之间存在关系，则使用双横杠 -- 和箭头 >, < 的组合来表示。

(a)-->(b) // 表示节点 a 到节点 b 的关系

(a)<--(b) // 表示节点 b 到节点 a 的关系

(a)--(b) // 允许不使用箭头，表示节点 a，b 之间存在关系，不考虑方向性

(a)-->(c)<--(b) // 表示节点 a，b 到节点 c 的关系

同时，Cypher 使用方括号 [] 和冒号 : 为关系赋予变量名与设置标签，使用管道符 | 隔开多个关系标签。

(a)-[r:WRITE]->(b) // 表示一个节点 a 到节点 b 的关系 r，标签为 WRITE

(a)-[r:WRITE|PARTICIPATE]->(b) // 表示一个节点 a 到节点 b 的关系 r，标签为 WRITE 或者 PARTICIPATE

#### 属性的表示

Cypher 中使用 Map 来表示节点和关系的属性。

(a:User {name: 'Gods\_Dusk', userId: 812252})-[:PARTICIPATE {rank: 2}]->(:Challenge {name: '第 23 期楼赛'}) // 节点 a 表示一位用户 Id 为 812252，用户名为 Gods\_Dusk，参加了第 23 期楼赛，名次为第二名

#### 路径的表示

Cypher 中**节点-关系-节点**是最基础的路径，可以用()--()表示， 对于较长的路径，Cypher 中使用星号 \*，数字和点号 . 来进行表示。

(a)-[\*2]->(b) // 表示节点 a，节点 b 之间的路径长度为 2，等价于 (a)-->()-->(b)

(a)-[\*3..5]->(b) // 表示节点 a，节点 b 之间的最小路径长度为 3，最大为 5

(a)-[\*3..]->(b) // 表示节点 a，节点 b 之间的路径长度至少为 3

(a)-[\*..5]->(b) // 表示节点 a，节点 b 之间的路径长度至多为 5

(a)-[\*]->(b) // 表示节点 a，节点 b 之间的路径长度为任意

## 环境配置

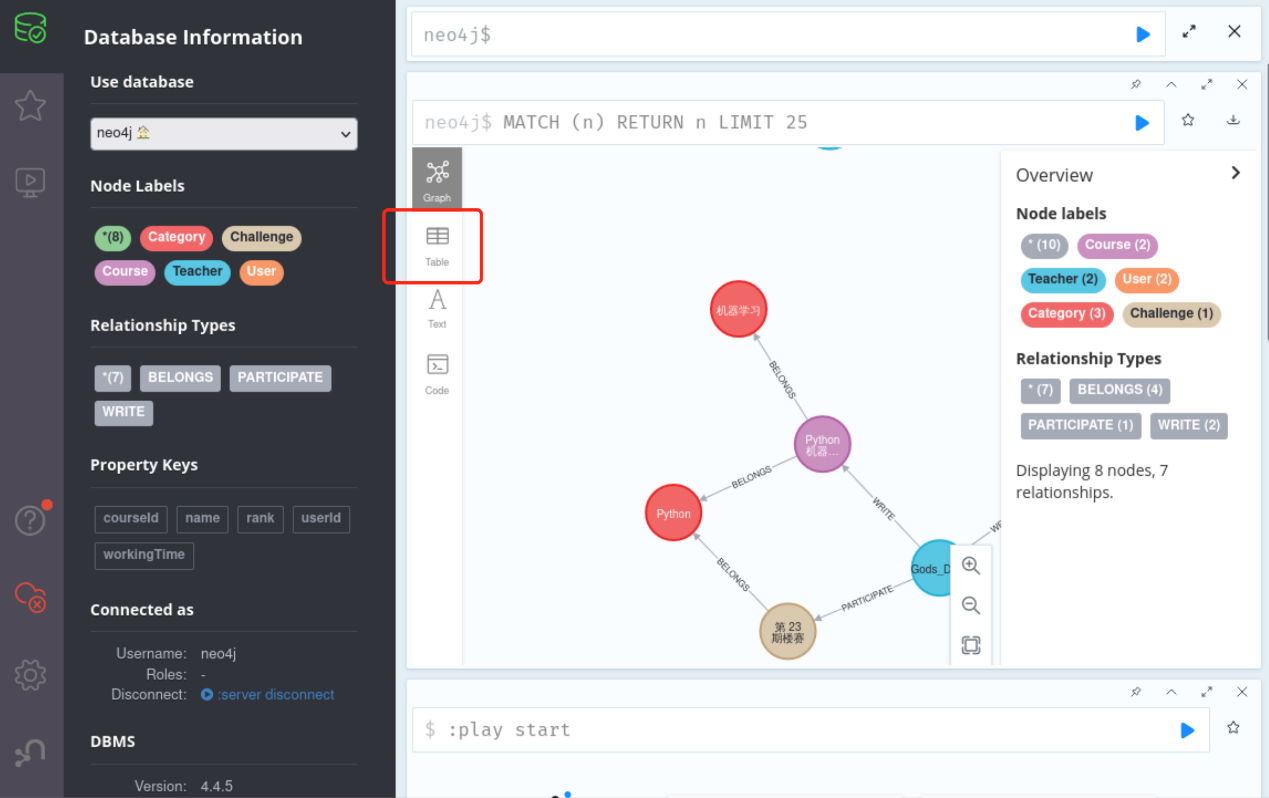
**若是从上一演练中进入，Neo4j 相关配置已保存，无需再次配置，可以直接跳过上一演练，进入后续的内容。**

#### Neo4j 数据导入与启动

（同上一演练，如果做过，可以忽略）

## Cypher 语句

打开浏览器，进入 Neo4j Browser，接下来的演练将在 Cypher 编辑器中完成，同时为了更明确地展示结果，结果栏中主要使用**表视图**进行运行结果的展示。



**由于使用快捷键复制粘贴有时候会将回车、单引号等符号转义，在 Cypher 编辑器中造成 Neo.ClientError.Statement.SyntaxError，因此遇到该错误时请检查复制粘贴内容是否正确。**

### Cypher 命名规范

* Cypher 中的关键字对大小写不敏感，官方推荐使用大写表示。
* 节点的标签推荐使用**大驼峰式命名法**，如 Course。
* 关系类型推荐使用**大写下划线命名法**，如 BELONGS\_TO。
* 属性推荐使用**小驼峰式命名法**，如 workingTime。
* 函数推荐使用**小驼峰式命名法**。

关于 Cypher 语句中的书写规范，如换行，空格等，可以参考官方提供的 [Cypher 风格指南](https://neo4j.com/docs/cypher-manual/current/styleguide/#cypher-styleguide)

### 基于图结构的查询

MATCH 命令用于图中的节点，关系，路径和属性的查询，RETURN 用于将结果返回。

因此基于图结构的查询语句的基本格式为：MATCH $图结构 RETURN $查询结果。

同时关键字 AS 用于为查询结果提供临时名称，增加了查询结果的可读性。

#### 查找所有节点

Cypher 语法中使用 (n) 表示节点，因此在 Cypher 编辑器中输入

MATCH (n)

RETURN n

就可以查询返回所有的节点。



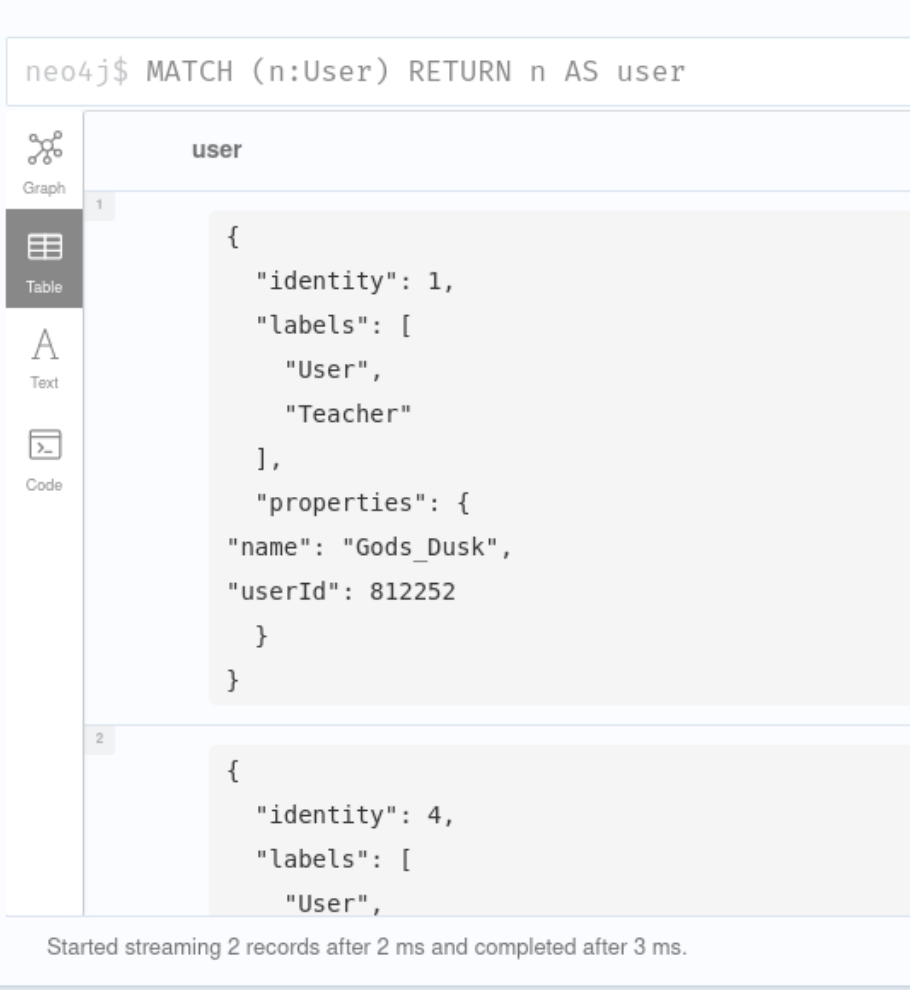
#### 获取指定标签和属性的节点

Cypher 编辑器中输入：

MATCH (n:User)

RETURN n AS user

获取到了标签为 **User** 的节点，同时重命名为 user 返回。



同理，若要获取标签为 **User**，属性 **name** 为 Gods\_Dusk 的节点，则可使用以下语句：

MATCH (n:User {name: 'Gods\_Dusk'})

RETURN n AS user

#### 关系获取

Cypher 语法中使用 [] 表示关系，但在图中不存在孤立的关系，即关系必然与节点同时存在，Cypher 编辑器中输入：

MATCH (n:User)-[r]->(m:Course)

RETURN r

就可以获得标签为 **User** 和 **Course** 之间所有的关系。



#### 路径获取

Cypher 语法中可以使用一个变量对查询到的路径进行接收并返回，Cypher 编辑器中输入：

MATCH p=()--()

RETURN p

就可以获得图数据中所有的长度为 1 的路径。

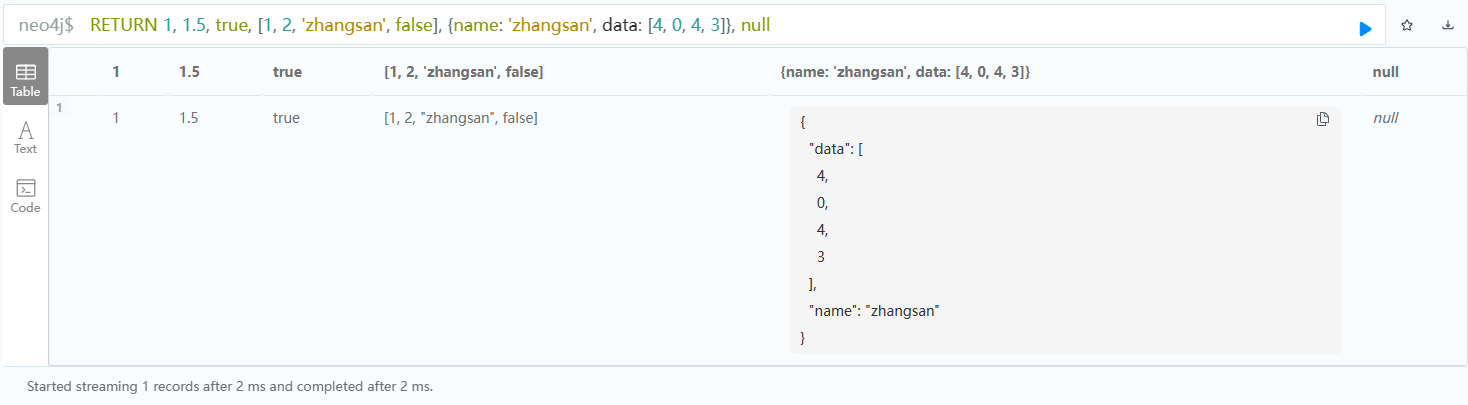


上图中路径查询返回格式为 [[{出节点属性}, {关系属性}, {入节点属性}], ...]。

### 数据类型与运算符

除了图相关的数据格式，在 Cyhper 语法中也支持常见数据格式如 List，Map，数字，字符串，布尔值。

RETURN 1, 1.5, true, [1, 2, 'zhangsan', false], {name: 'zhangsan', data: [4, 0, 4, 3]}, null



#### 基本操作符

* 等于 =
* 不等于 <>

基本操作符适用于所有数据格式。

RETURN 1=1, 1<>1, 1=null, '1'=1, null=null



#### List 操作

Cypher 中 List 支持的操作有：

* 拼接 +
* 判断判断元素是否存在 IN
* 切片操作 []

RETURN [1, 2] + [3], [1, 2] + 3 + 4, 'zhang' IN ['ZHANG', 'SAN'], [1, 2, 3, 4][3], [1, 2, 3, 4][1..3]

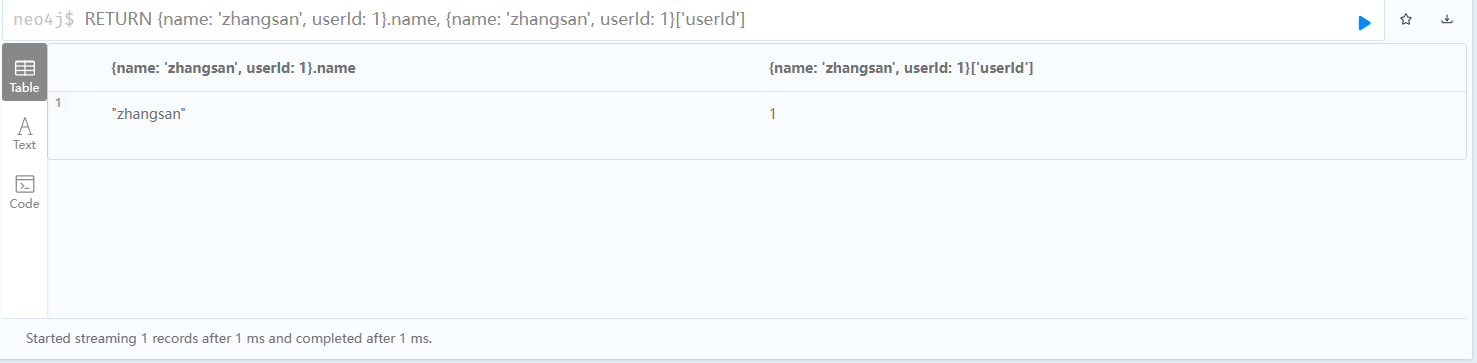


#### Map 操作

Cypher 中 Map 支持的操作有：

* 通过键获取值 . 和 []

RETURN {name: 'zhangsan', userId: 1}.name, {name: 'zhangsan', userId: 1}['userId']



在图结构查询中，查询返回的节点与关系也支持 Map 操作。

MATCH (n:User) RETURN n.name, n['name']



上述语句查询了所有 **User** 节点并以两种方式返回其 **name** 属性。

#### 数学运算

Cypher 数学运算包括以下操作符：

* 加法 +
* 减法 -
* 乘法 \*
* 除法 /
* 取余 %
* 幂次 ^

MATCH (n:User {name: '云课管理员'})

RETURN n.userId + 2 \* 3 / 4 % 5 ^ 6, 1 + 2 \* 3 / 4 % 5 ^ 6

上述查询语句先查询了 **name** 为**云课管理员**的 **User** 节点，返回其 **userId** 属性，并加上 2 \* 3 / 4 % 5 ^ 6 的值。因为该节点的 **userId** 值为 1， 所以与 1 + 2 \* 3 / 4 % 5 ^ 6 所计算的结果一致。

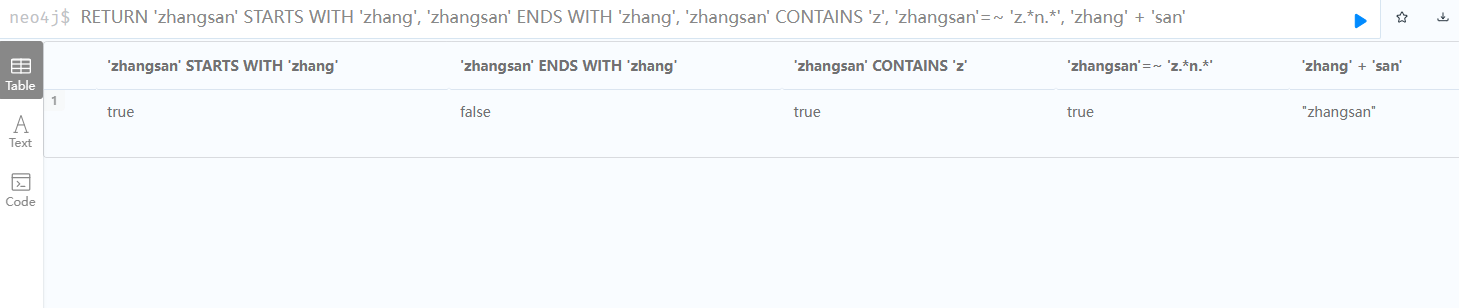


##### 字符串运算

Cypher 字符串运算包括以下关键字：

* 首字符串匹配 STARTS WITH
* 尾字符串匹配 ENDS WITH
* 内部字符串匹配 CONTAINS
* 正则匹配 =~
* 字符串拼接 +

RETURN 'zhangsan' STARTS WITH 'zhang', 'zhangsan' ENDS WITH 'zhang', 'zhangsan' CONTAINS 'z', 'zhangsan'=~ 'z.\*n.\*', 'zhang' + 'san'

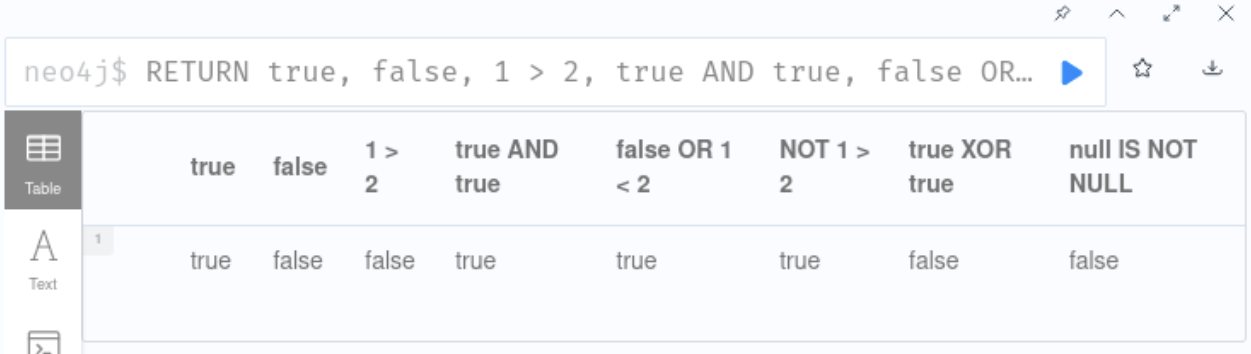


#### 比较与布尔运算

Cypher 比较运算和布尔包括以下操作符：

* 大于 >
* 等于 =
* 小于 <
* 大于等于 >=
* 小于等于 <=
* 与操作 AND
* 或操作 OR
* 非操作 NOT
* 异或操作 XOR
* null 判断 IS NULL, IS NOT NULL

RETURN true, false, 1 > 2, true AND true, false OR 1 < 2, NOT 1 > 2, true XOR true, null IS NOT NULL



### 查询过滤

Cypher 中提供了一系列关键字来对查找结果进行过滤。

#### 条件查找

Cypher 提供 WHERE 关键字进行条件查找符合要求的结果。

MATCH (n:User)-->(m:Course)

WHERE n.name = 'Gods\_Dusk'

RETURN m.name

上述语句查询了与用户 'Gods\_Dusk' 相关的所有课程并返回课程名称。



等价于语句：

MATCH (n:User {name: 'Gods\_Dusk'})-->(m:Course)

RETURN m.name



但与第二条语句不同的是，WHERE 关键字支持更加复杂的逻辑查询，如：

MATCH (n:User)-->(m:Course)

WHERE n.name IN ['Gods\_Dusk', '云课管理员'] AND n.userId > 10 OR NOT n.name STARTS WITH '云课'

RETURN m.name

上述语句的查询条件是 **name** 属性包含在 **Gods\_Dusk** 和**云课管理员**中，且 **userId** 大于 **10**，或者 **name** 属性的起始字符串不为**云课**。



#### 排序

关键字 ORDER BY 用于给查询结果排序，同时可以结合升序 ASC 和降序 DESC 使用。

MATCH (n:User)

RETURN n.name ORDER BY n.userId DESC

上述语句对 **User** 节点以 **userID** 降序的方式返回 **name** 属性。



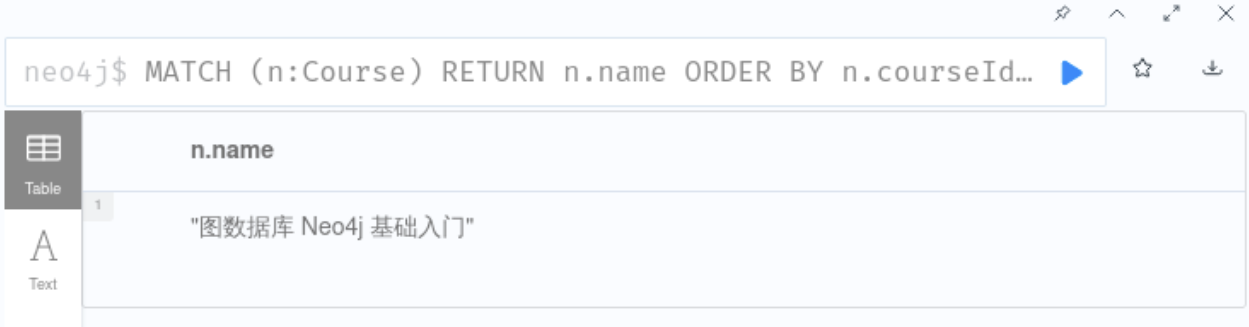
#### 筛选

若要对返回的结果进行筛选，则可以使用 LIMIT 限制返回数量，SKIP 限制起始位置。

MATCH (n:Course)

RETURN n.name ORDER BY n.courseId SKIP 1 LIMIT 1

上述语句使用 **courseId** 对 **Course** 节点进行排序，然后跳过第一个结果，从第二个结果开始，返回一个结果。



### 函数

Cypher 提供的函数种类丰富，在本演练中主要联系其中一部分常见的函数。

#### 标量函数

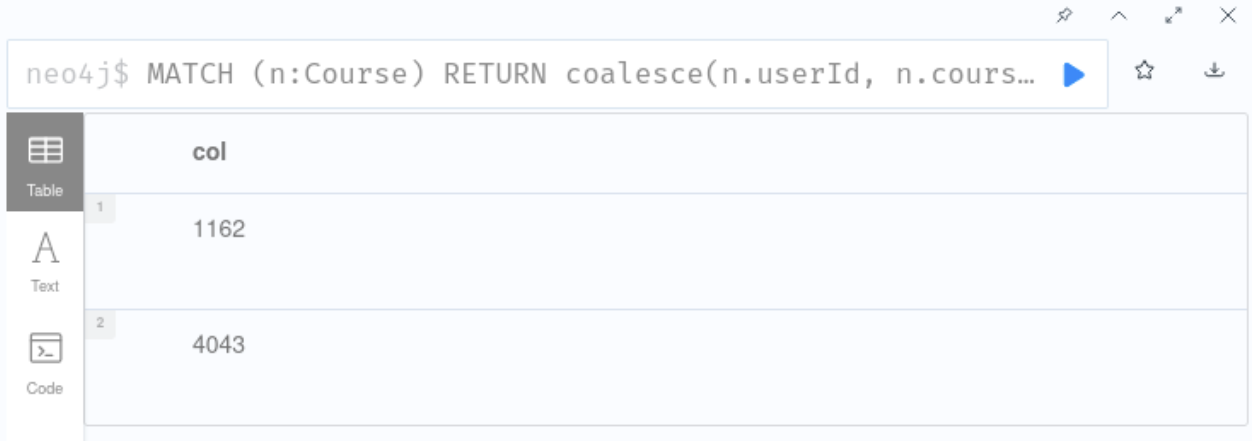
标量函数，接收单个或者多个输入，返回单个值。

* 函数 coalesce() 返回输入的第一个非空值，当所有的都为空时，则返回 null。

MATCH (n:Course)

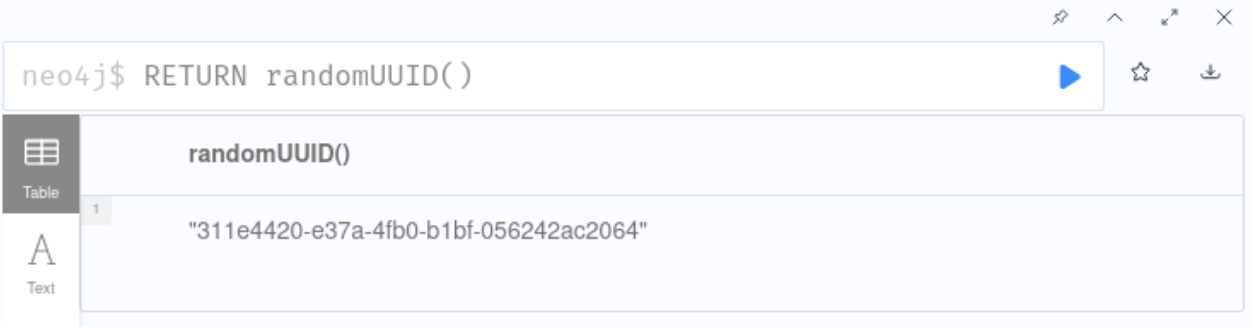
RETURN coalesce(n.userId, n.courseId) AS col

**Course** 节点不存在 **userId** 属性，因此返回的是 **courseId** 的值。



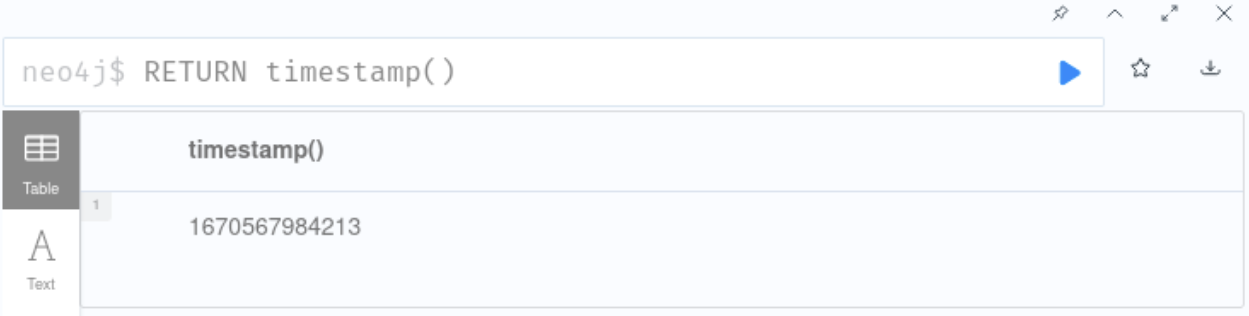
* 随机生成函数 randomUUID() 返回一个随机生成的通用唯一识别码 (Universally Unique Identifier)。

RETURN randomUUID()



* 时间戳函数 timestamp() 返回当前时间戳，单位为毫秒。

RETURN timestamp()



* 主键获取函数 id() 返回节点或者关系的内置主键，即在**图视图**中点击节点或关系时，下边栏显示的 <id> 属性。

MATCH (n:User {name: 'Gods\_Dusk'})-[r]->(m:Course {courseId: 4043})

RETURN id(n), id(m), id(r)



* 属性获取函数 properties() 以 Map 的形式返回节点和关系的属性。

MATCH (n:User)-[r]-()

RETURN properties(n), properties(r)



* 函数 size() 可以计算 List 与字符串的长度，count{}用于计算路径的数量。

MATCH (n:User)

RETURN n.name, size(n.name), size([n.name, n.userId]), count{(n)-->()-->()}

第一个 size() 用于计算 **n.name** 的字符串长度，第二个 size() 用于计算 List 的长度，第三个 count{} 用于计算路径 (n)-->()-->() 的数量

注意：一个中文字符和一个英文字符（数字和\_），都是1个字符的长度。

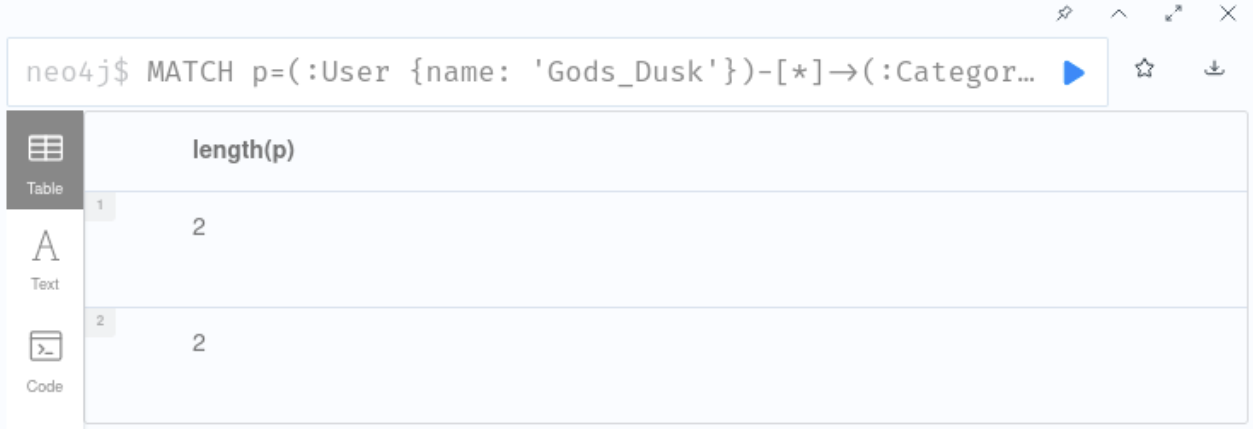


* 函数 length() 计算一个路径的长度。

MATCH p=(:User {name: 'Gods\_Dusk'})-[\*]->(:Category {name: 'Python'})

RETURN length(p)

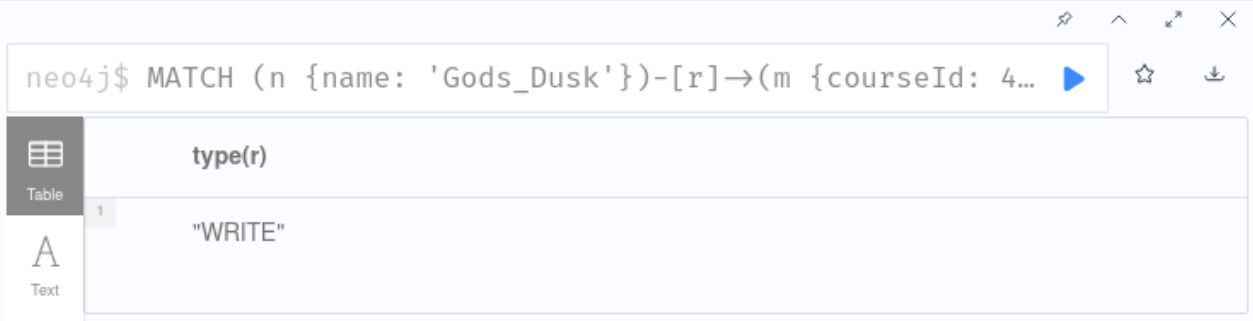
节点 Gods\_Dusk 到节点 Python 一共有两条路径，分别是 ({name 'Gods\_Dusk'})->[:WRITE]->(:Course)->[:BELONGS]->({name 'Python'}) 和 ({name 'Gods\_Dusk')->[:PARTICIPATE]->(:Challenge)->[:BELONGS]->({name 'Python'})，长度均为 2。



* 类型获取函数type() 接收关系，返回其类型。

MATCH (n {name: 'Gods\_Dusk'})-[r]->(m {courseId: 4043})

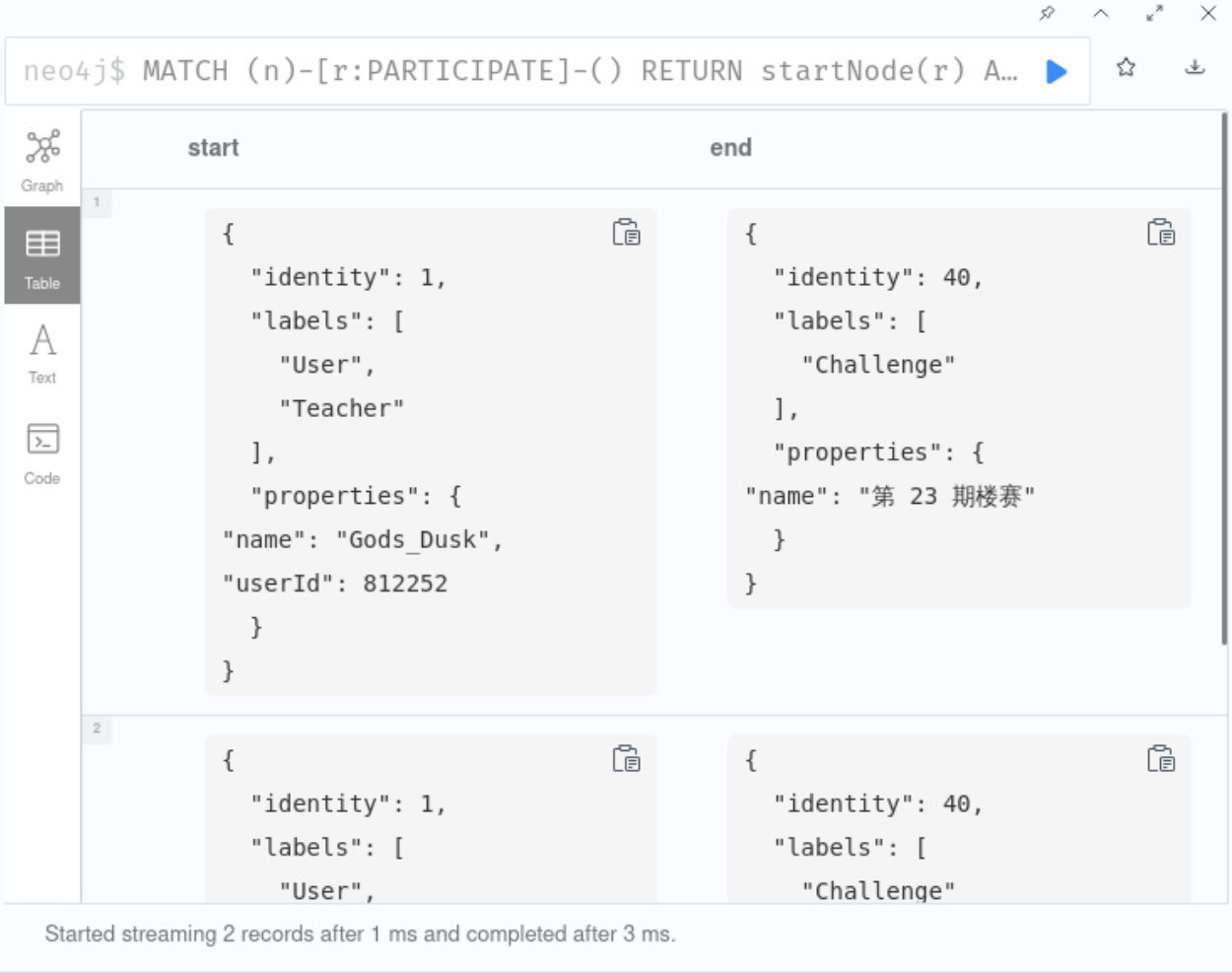
RETURN type(r)



* 关系出入节点获取函数 startNode()，endNode() 分别返回一个关系的出节点和入节点。

MATCH (n)-[r:PARTICIPATE]-()

RETURN startNode(r) AS start, endNode(r) AS end



* 字符串转换函数 toBoolean()，toFloat()，toInteger() 分别将字符串转换为布尔值，浮点型和整型，当输入字符串格式非法时返回 null。

RETURN toBoolean('True'), toBoolean('true'), toFloat('123.3'), toInteger('23'), toInteger('a')



* List 首尾取值函数 head()，last() 分别返回 List 第一个和最后一个元素，当输入为 null 时返回 null。

RETURN head([1, 2, 3, 4]), head(null), last([1, 2, 3, 4]), last(null)



#### 列表函数

列表函数接收单个或者多个输入，返回一个 List。

* 属性名获取函数 keys() 获取节点，关系和 Map 的属性名。

MATCH (n)-[r:PARTICIPATE]->()

RETURN n.name, keys(n), keys(r)



* 标签获取函数 labels() 获取一个节点的所有标签。

MATCH (n)

RETURN n.name, labels(n)



* nodes() 和 relationships() 函数分别获取一个路径中的节点和关系

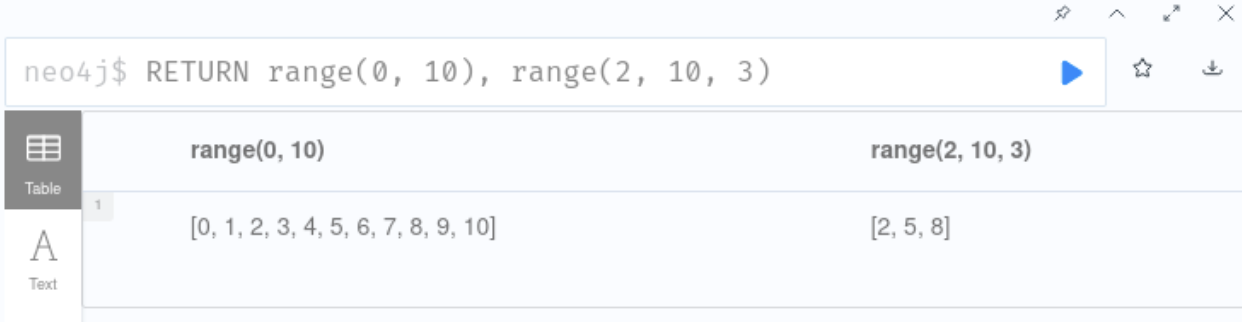
MATCH p=()-[\*2]->()

RETURN nodes(p), relationships(p)



* 函数 range() 创建一个整数列表，用法为 range(start, end [, step=1])

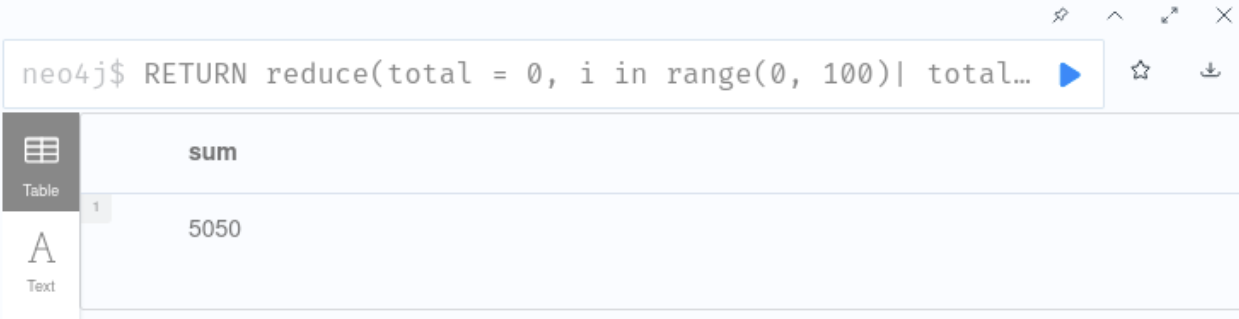
RETURN range(0, 10), range(2, 10, 3)



* 函数 reduce() 应用在列表上，对列表中的每个元素进行迭代计算，在元素上运行表达式，把当前的结果存储在累加器中，进行迭代计算，并返回最终计算的标量结果，标准语法为：reduce(accumulator = initial, variable IN list | expression)。

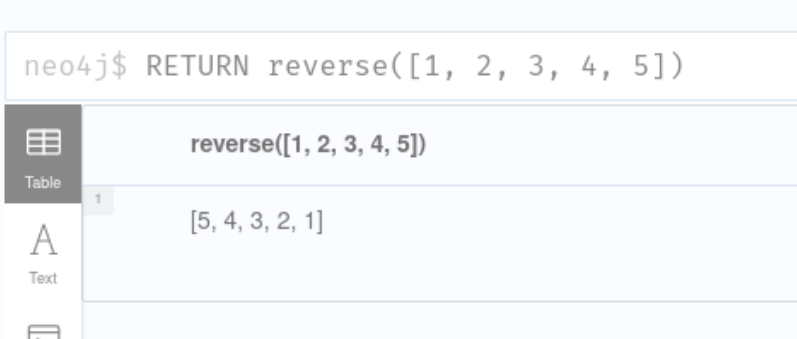
RETURN reduce(total = 0, i in range(0, 100)| total + i) AS sum

上述语句计算了 0 到 100 的和。



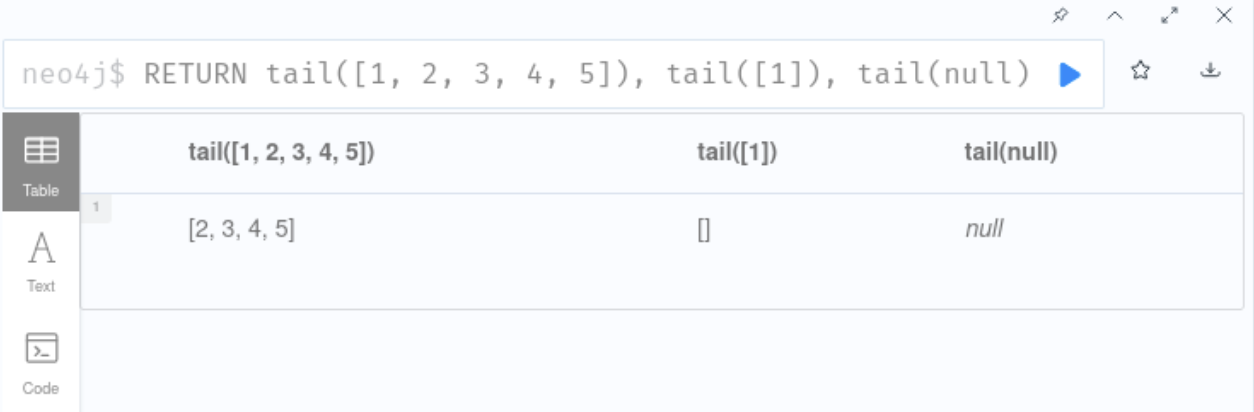
* 函数 reverse() 在输入格式为 List 时，用于反转 List。

RETURN reverse([1, 2, 3, 4, 5])



* 函数 tail() 接收一个 List 返回其从第二个元素开始的 List，当输入为 null 时返回 null。

RETURN tail([1, 2, 3, 4, 5]), tail([1]), tail(null)



#### 数学函数

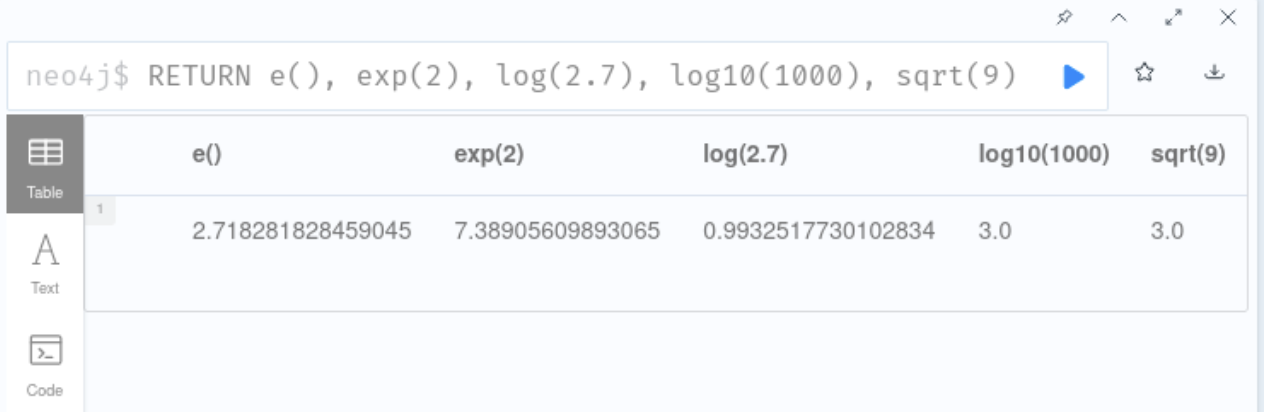
* 数值函数包括：绝对值函数 abs()；向上取整函数 ceil()；向下取整函数 floor()；随机数生成函数 rand()；四舍五入函数 round()；符号函数 sign()。

RETURN abs(-1.5), ceil(1.2), floor(1.8), rand(), round(3.4), sign(-10)



* 对数指数函数包括：欧拉数 e()；欧拉数为底的指数函数 exp()；自然对数函数 log()；以 10 为底的对数函数 log10()；平方根函数 sqrt()。

RETURN e(), exp(2), log(2.7), log10(1000), sqrt(9)



* 三角函数包括：反余弦函数 acos()；反正弦函数 asin()；反正切函数 atan()；反余切函数 atan2()；余弦函数 cos()；余切函数 cot()；弧度转换函数 degrees()；圆周率 pi()；角度转换函数 radians()；正弦函数 sin()；正切函数 tan()。

RETURN acos(cos(pi())) AS ans1, sin(asin(-1)) AS ans2, tan(atan(1)) AS ans3, atan2(1, -1) \* 180 / pi() AS ans4, cot(45 \* pi() / 180) AS ans5, degrees(pi()) AS ans6, radians(180) AS ans7



#### 字符串函数

* 字符串分割函数 split()，接收字符串和一个分隔符，返回分割后的 List。

MATCH (n:User)

RETURN split(n.name, '\_')



* 函数 reverse() 在输入格式为字符串时，用于反转字符串。

MATCH (n:Course)

RETURN reverse(n.name)



* 函数 replace() 用于字符串替换。

MATCH (n)

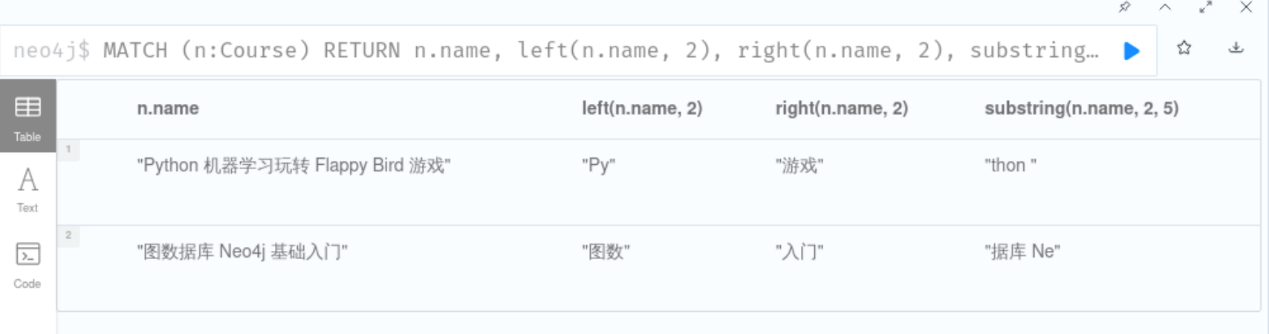
RETURN replace(n.name, ' ', '\*')



* 字符串截取函数：左截取函数 left()；右截取函数 right()；中间截取函数 substring()。

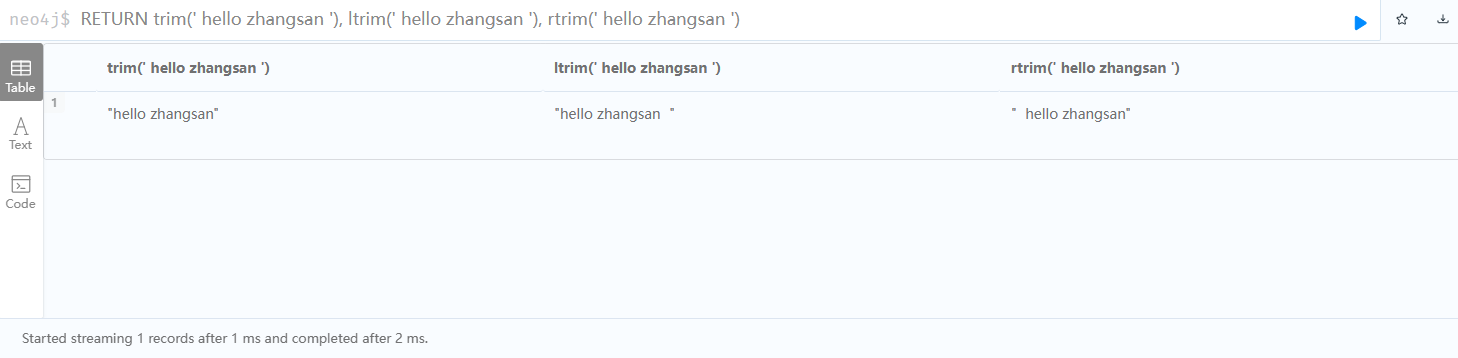
MATCH (n:Course)

RETURN n.name, left(n.name, 2), right(n.name, 2), substring(n.name, 2, 5)



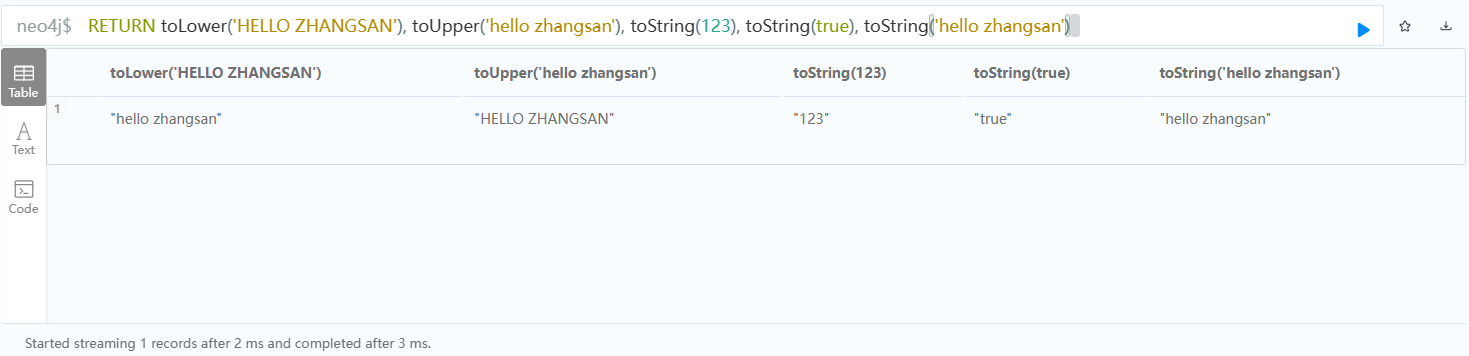
* 字符串过滤函数：全过滤函数 trim()，左过滤函数 ltrim()，右过滤函数 rtrim()。 用于过滤指定的字符串，最常见的用途是移除字首或字尾的空白。

RETURN trim(' hello zhangsan '), ltrim(' hello zhangsan '), rtrim(' hello zhangsan ')



* 字符串转换函数包括：小写转换函数 toLower()，大写转换函数 toUpper()，字符串类型转换函数 toString() 。

RETURN toLower('HELLO ZHANGSAN'), toUpper('hello zhangsan'), toString(123), toString(true), toString('hello zhangsan')



#### 聚合函数

聚合函数对一组值执行计算，并返回单个值。

* 函数 collect() 用于将查询结果压缩到一个 List 中，若输入为 null，则返回空的 List。

MATCH (n:User)

RETURN collect(n.name), collect(null)



* 函数 avg() 用于求平均值。

MATCH ()-[r:WRITE]->()

RETURN collect(r.workingTime), avg(toInteger(split(r.workingTime, ' ')[0]))

上述语句用于计算 **Write** 关系的平均 **workingTime** 值。

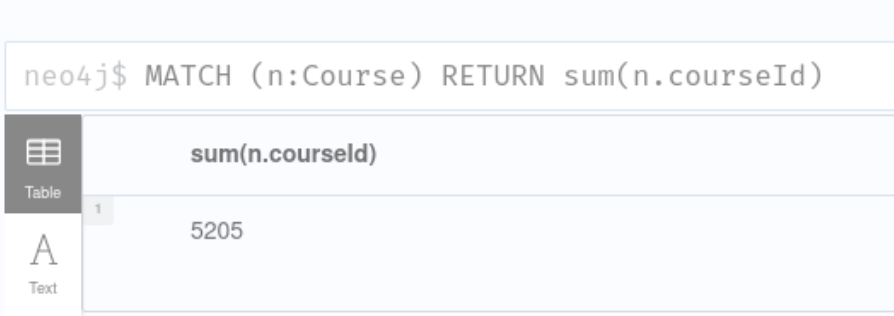


* 函数 sum() 用于求和。

MATCH (n:Course)

RETURN sum(n.courseId)

上述语句用于查询计算 **Couser** 节点的 **couserId** 之和。

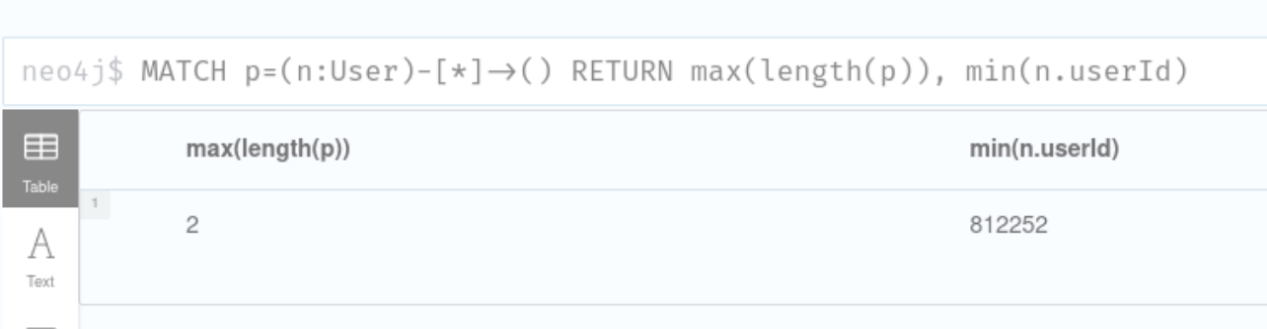


* 极值函数 max()， min() 分别用于计算最大值和最小值。

MATCH p=(n:User)-[\*]->()

RETURN max(length(p)), min(n.userId)

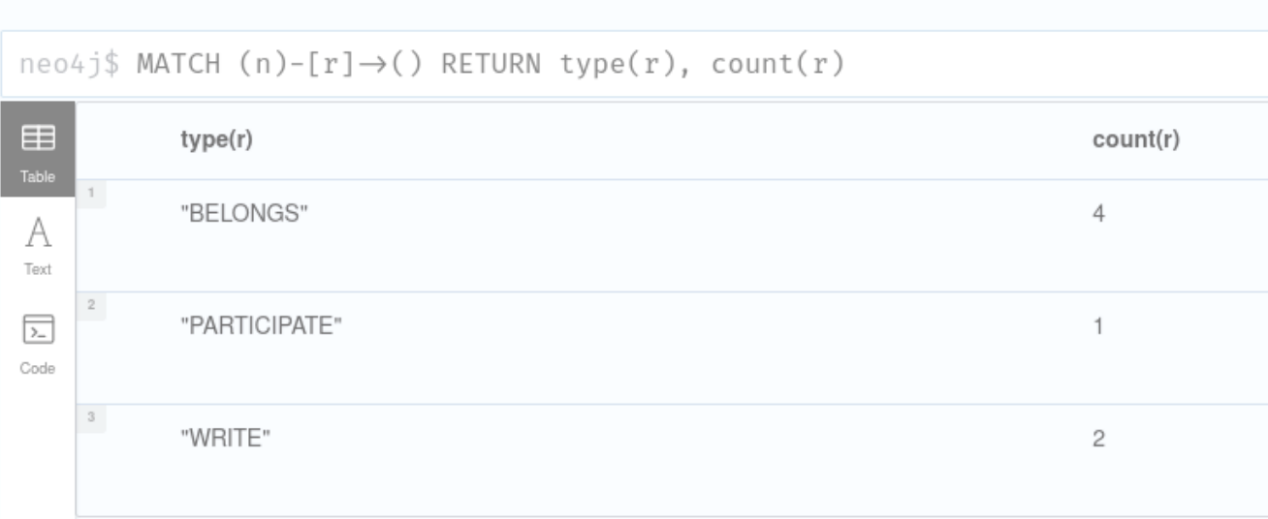
上述语句用于查询计算带有 **User** 节点的路径的长度最大值和 **userId** 的最小值



* 函数 count() 用于统计查询结果数量。

MATCH (n)-[r]->()

RETURN type(r), count(r)



上述语句统计了每一种关系类型的数量。

* 方差函数 stDev()， stDevP() 分别用于计算无偏样本方差和有偏样本方差。

MATCH p=()-[\*]->()

RETURN stDev(length(p)), stDevP(length(p))

上述语句用于查询计算图中所有路径长度的方差。

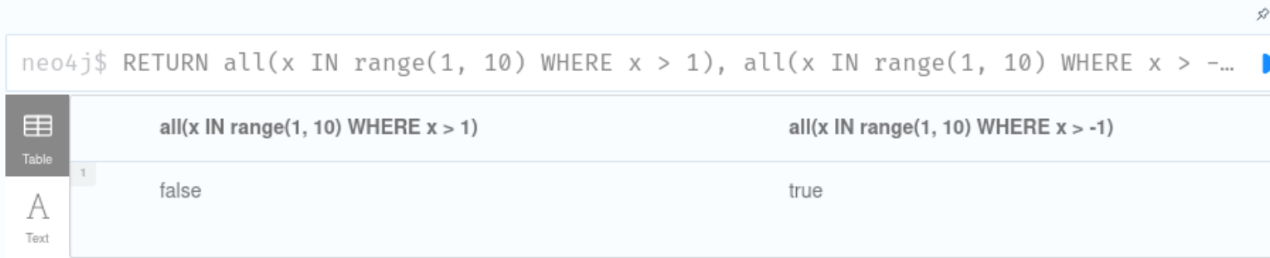


#### 断言函数

断言函数接受一个表达式，并返回该表达式的布尔值。

* 函数 all() 判断列表中的所有元素是否同时满足要求，基本语法为 all(variable IN list WHERE predicate)

RETURN all(x IN range(1, 10) WHERE x > 1), all(x IN range(1, 10) WHERE x > -1)



* 函数 none() 判断列表中的元素是否同时均不满足要求，基本语法为 none(variable IN list WHERE predicate)

RETURN none(x IN range(1, 10) WHERE x > 5), none(x IN range(1, 10) WHERE x < 1)

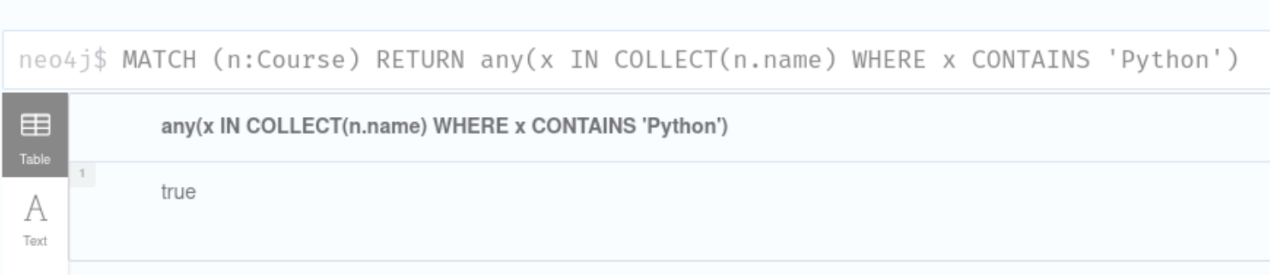


* 函数 any() 判断列表中的所有元素是否至少有一项满足要求，基本语法为 any(variable IN list WHERE predicate)

MATCH (n:Course)

RETURN any(x IN COLLECT(n.name) WHERE x CONTAINS 'Python')

上述语句判断了是否存在带有 Python 关键字的 **Course** 节点。

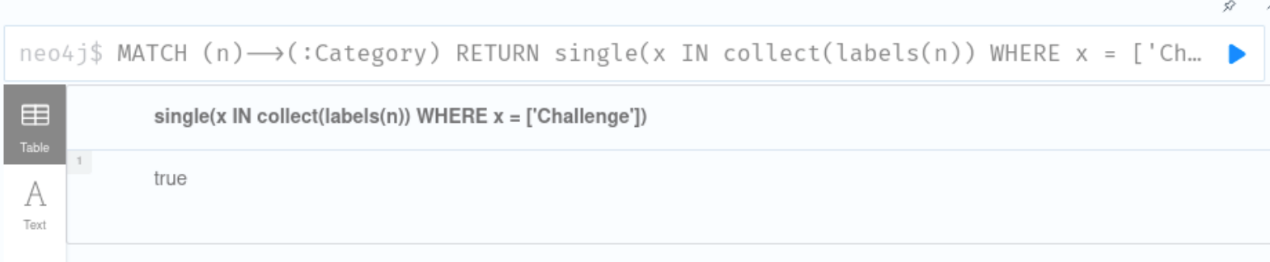


* 函数 single() 判断列表中的所有元素是有且只有一项满足要求，基本语法为 single(variable IN list WHERE predicate)

MATCH (n)-->(:Category)

RETURN single(x IN collect(labels(n)) WHERE x = ['Challenge'])

上述语句先搜索了以 **Category** 为入节点，长度为 1 的路径，然后再判断所有出节点的标签是否有且仅有一项为 ['Challenge']，要注意到 labels() 函数返回数据类型是 List。

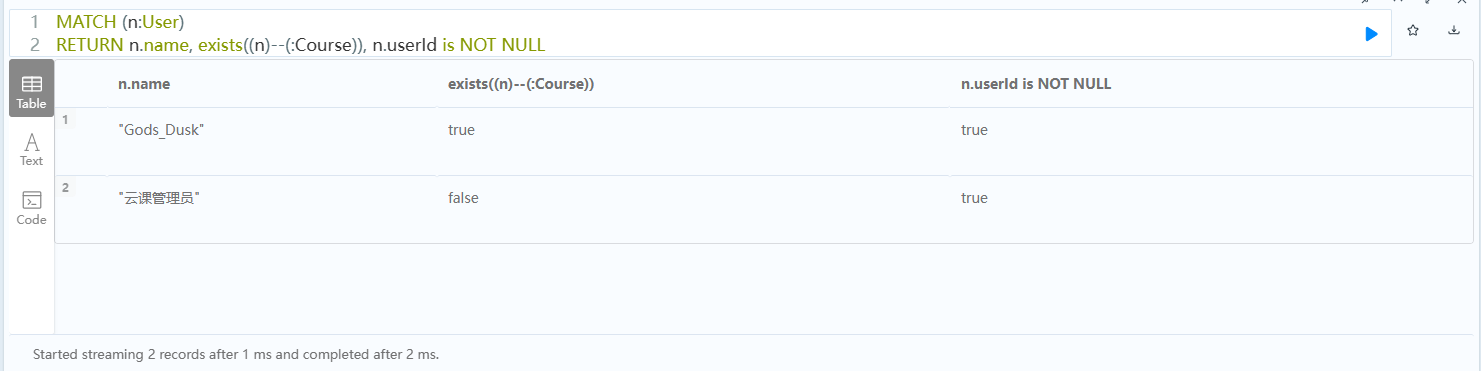


* 函数 exists() 用于判断是否存在某个路径，基本语法为exists(pattern-or-property)。
* 注意：判断变量的某个属性是否存在，使用：variable.property IS NOT NULL。

MATCH (n:User)

RETURN n.name, exists((n)--(:Course)), n.userId is NOT NULL

上述语句用于查询节点 **User** 是否存在 **Course** 节点与其相连以及是否存在 **userId** 属性。



### 复杂查询

#### 查询链接

WITH 允许将查询子句链接在一起，将前面部分的结果输送到下一个部分中作为起点或条件。

MATCH (n:User)-->(m:Course) WITH m AS course

MATCH (course)-->(k:Category)

RETURN k.name

该语句先查询了 **User** 节点对应的所有 **Course** 节点，再二次查询 **Course** 节点对应的所有 **Category** 节点。

#### 可选匹配

可选匹配 OPTIONAL MATCH 子句用于匹配搜索模式中描述的匹配项，与 MATCH 不同的是，MATCH 没有搜索到匹配结果，不会返回任何记录，如 Cypher 语句：

MATCH (n:User {name: '云课管理员'})-->(m:Course)

RETURN m



而 OPTIONAL MATCH 会返回 null。

MATCH (n:User {name: '云课管理员'})

OPTIONAL MATCH (n)-->(m:Course)

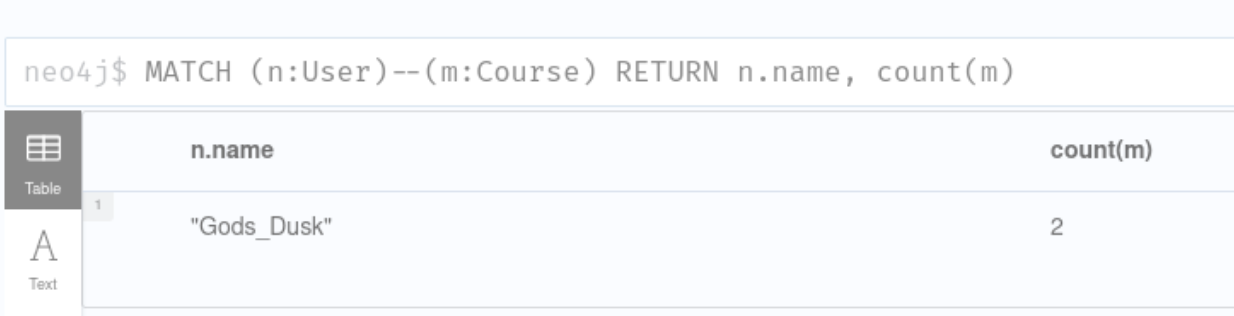
RETURN m, m.name



因此，可以很方便的对 null 值进行下一步处理，比如要统计每一个 User 相连的 Course 节点数量，只使用 MATCH 无法输出数量为 0 的节点。

MATCH (n:User)--(m:Course)

RETURN n.name, count(m)

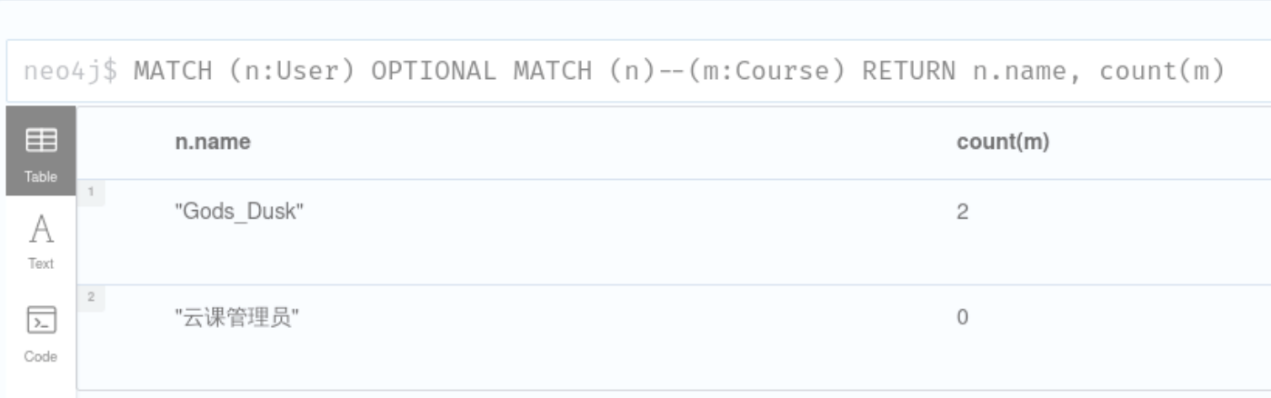


OPTIONAL MATCH 则可以输出这些结果。

MATCH (n:User)

OPTIONAL MATCH (n)--(m:Course)

RETURN n.name, count(m)



#### 分支语句

Cypher 中的分支语句的关键字为 CASE，使用方法有两种：Simple CASE form、Generic CASE form。

* Simple CASE form 的语法为 CASE test WHEN value THEN result [WHEN ...] [ELSE default] END。
* Generic CASE form 的语法为 CASE WHEN predicate THEN result [WHEN ...] [ELSE default] END。

Simple CASE form 的语法等效于 IF-ELSE 语句，Cypher 计算每个 WHEN 子句中的布尔条件，并且只执行条件为真的第一个语句块。仅当所有 WHEN 子句条件均为假时，才会执行末尾的 ELSE 子句（如果有）。如果所有 WHEN 子句的布尔条件都为假，且没有 ELSE 语句，则返回 null。

而 Generic CASE form 的语法等效于 SWITCH 语句。

例如，需要输出所有节点的 userId 属性，如果属性不存在，则输出-1。

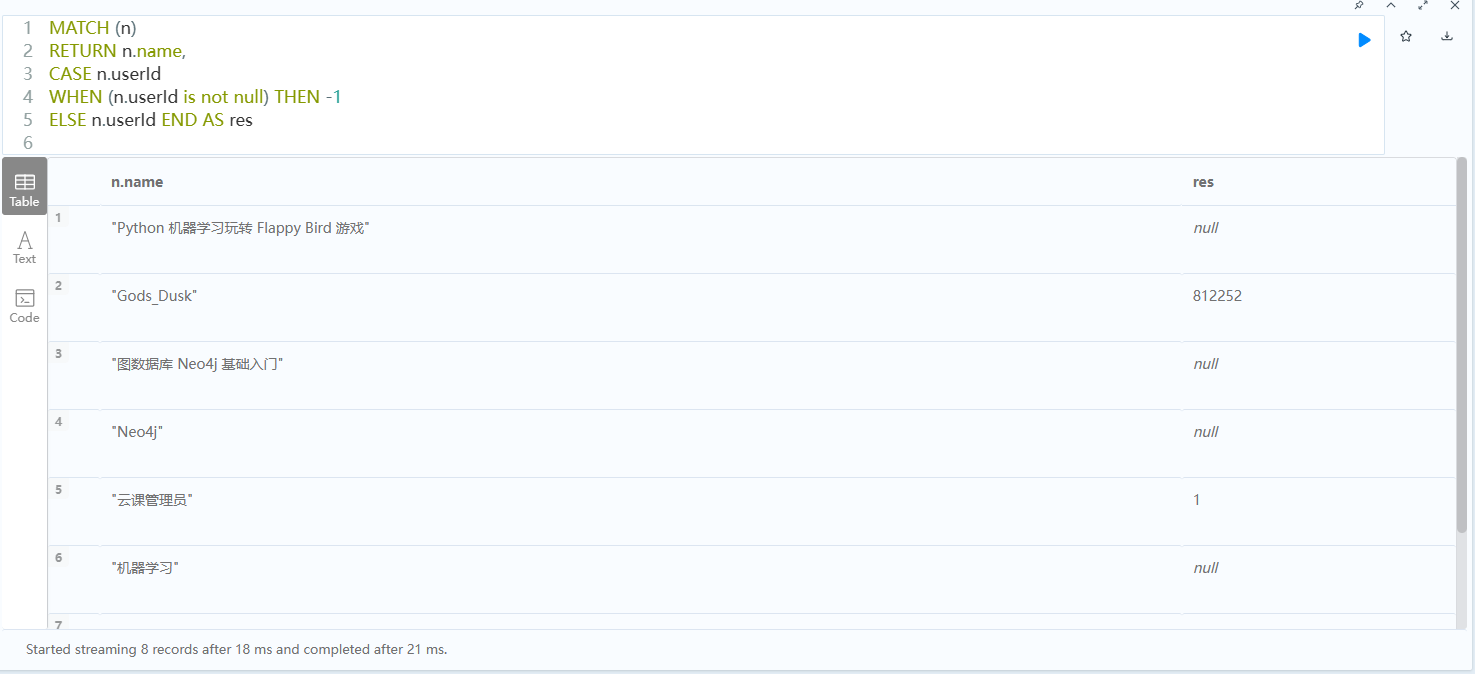
MATCH (n)

RETURN n.name,

CASE n.userId

WHEN (n.userId is not null) THEN -1

ELSE n.userId END AS res



上述 Simple CASE form 方法无法执行该功能，而 Generic CASE form 可以返回正确的结果。

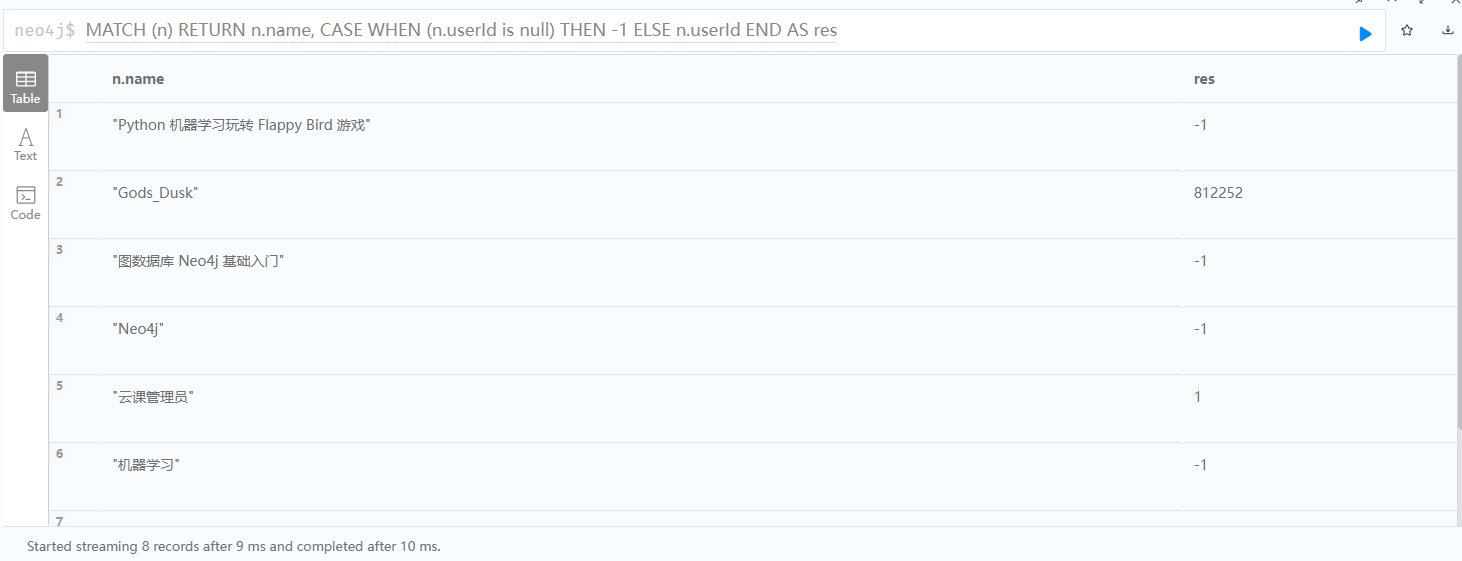
MATCH (n)

RETURN n.name,

CASE

WHEN (n.userId is null) THEN -1

ELSE n.userId END AS res



#### 查询集合并

UNION 用于合并两个查询集，合并不含重复结果。

MATCH (n:User)

RETURN n.userId AS id

UNION

MATCH (n:Course)

RETURN n.courseId AS id



UNION ALL 合并包含重复结果。

MATCH (n:User)

RETURN n.name

UNION ALL

MATCH (n)

RETURN n.name



在这里列名必须一致，否则 Cypher 会提示错误。

#### 列表拆解

UNWIND 用于将列表拆解，相当于函数 collect() 的逆操作，UNWIND 结合其他语法可以写出许多复杂的查询。 最简单的用法为：

UNWIND [1,2,3] AS x

RETURN x



#### 去重

DISTINCT 用于对返回数据的去重。

UNWIND [1,1,2,2,3,4] AS x

RETURN DISTINCT x



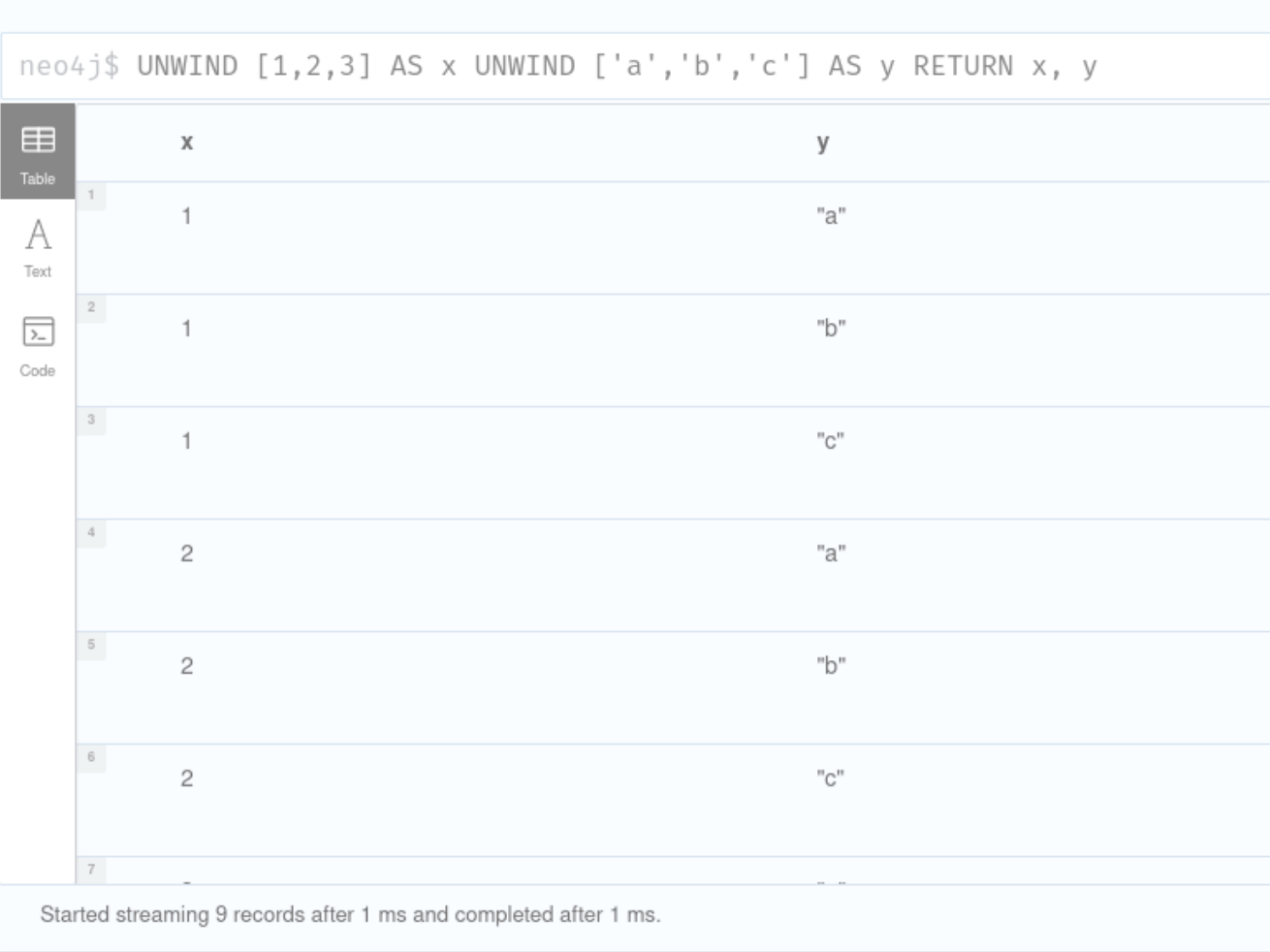
#### 笛卡儿积

笛卡尔积是将两个集合中的元素两两组合形成的新的集合，笛卡尔积是部分数据库的一个重要理论基础。

UNWIND [1,2,3] AS x

UNWIND ['a','b','c'] AS y

RETURN x, y



UNWIND 展开两个列表其结果将为笛卡尔积，因此上述语句会返回 9 条记录，同理多个 WITH 语句：

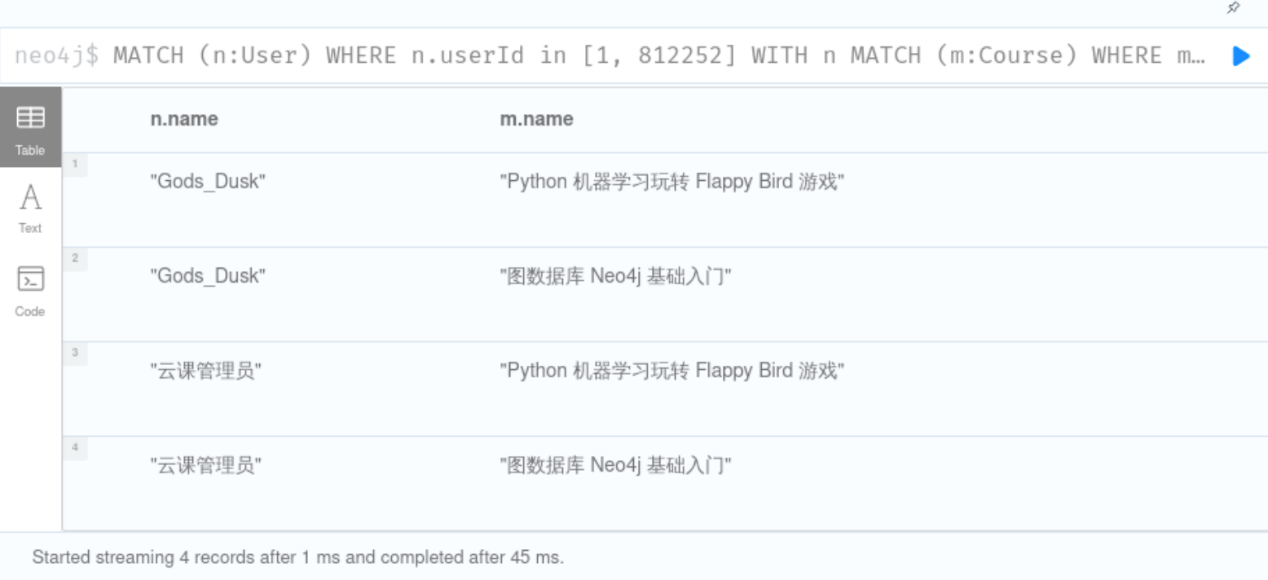
MATCH (n:User)

WHERE n.userId in [1, 812252] WITH n

MATCH (m:Course)

WHERE m.courseId in [4043, 1162] WITH n, m

RETURN n.name, m.name



若在 MATCH 语句中一次查询多个变量，也会进行笛卡儿积操作。

MATCH (n:User), (m:User)

RETURN n.name



#### 最短路径

图数据库最大的特点之一就是可以快速地在图中查询两个节点的最短路径。

查询单个最短路径使用 shortestpath() 函数。

MATCH (n:Category {name: '机器学习'})

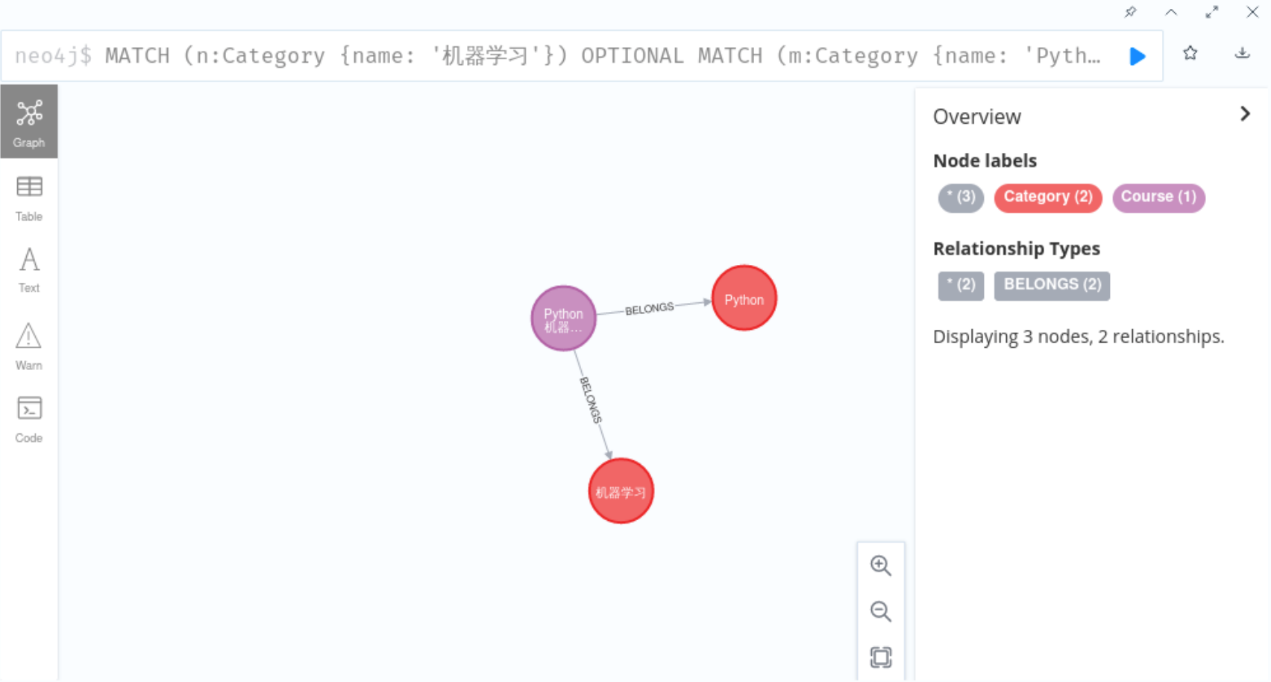
OPTIONAL MATCH (m:Category {name: 'Python'})

RETURN shortestPath((n)-[\*]-(m)) AS p

上述语句查询了**机器学习**节点到 **Python** 节点的最短路径，为 ({name: '机器学习'})<-[:BELONGS]-({name: 'Python 机器学习玩转 Flappy Bird 游戏'})-[::BELONGS]->({name: 'Python'})。



在**图示图**中结果展示如下：



最短路径同时也可以指定关系类型与路径长度。

MATCH (n:User {name: 'Gods\_Dusk'})

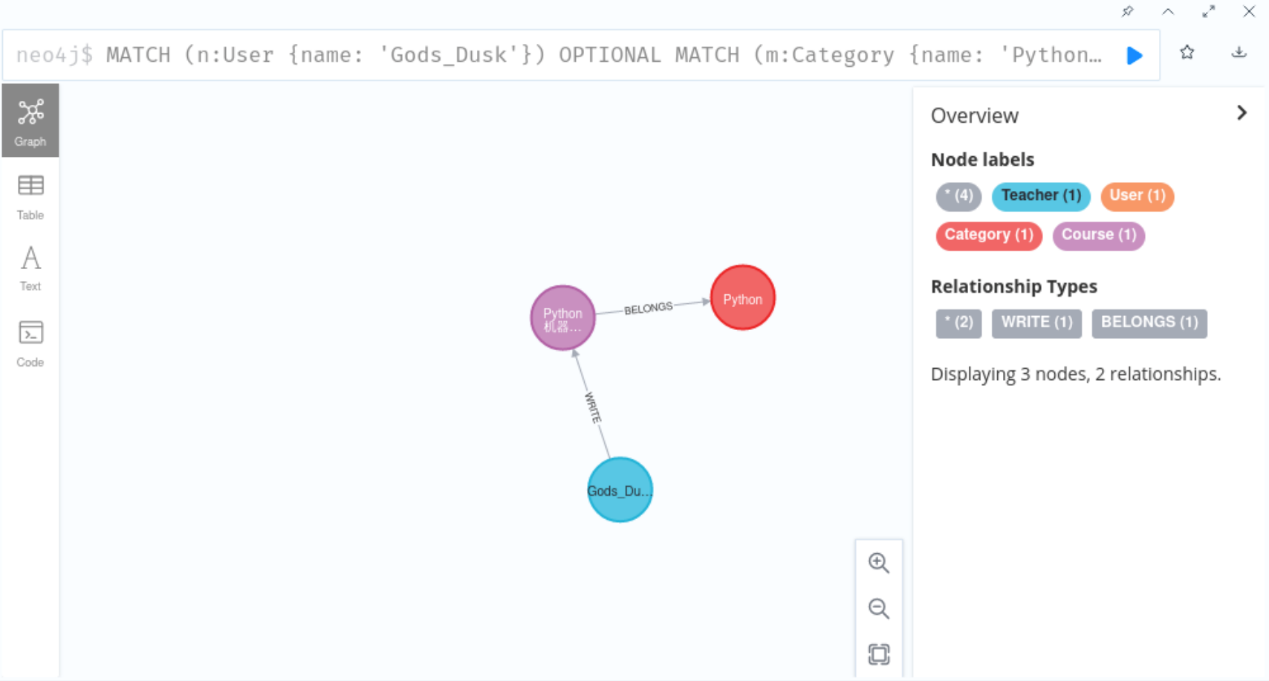
OPTIONAL MATCH (m:Category {name: 'Python'})

RETURN shortestPath((n)-[:WRITE|BELONGS\*..10]->(m)) AS p

上述语句查询了 **Gods\_Dusk** 节点到 **Python** 节点，只经过 **WRITE** 或 **BELONGS** 类型关系且长度不超过 10 的路径。



在**图示图**中结果展示如下：



shortestpath() 只返回两个节点之间单个最短路径，而函数 allShortestPaths() 返回所有的最短路径。

MATCH (p:Category) WITH collect(p) as nodes

UNWIND nodes AS source

UNWIND nodes AS target

WITH source, target WHERE id(source)<>id(target)

MATCH paths = allShortestPaths((source)-[\*..10]-(target))

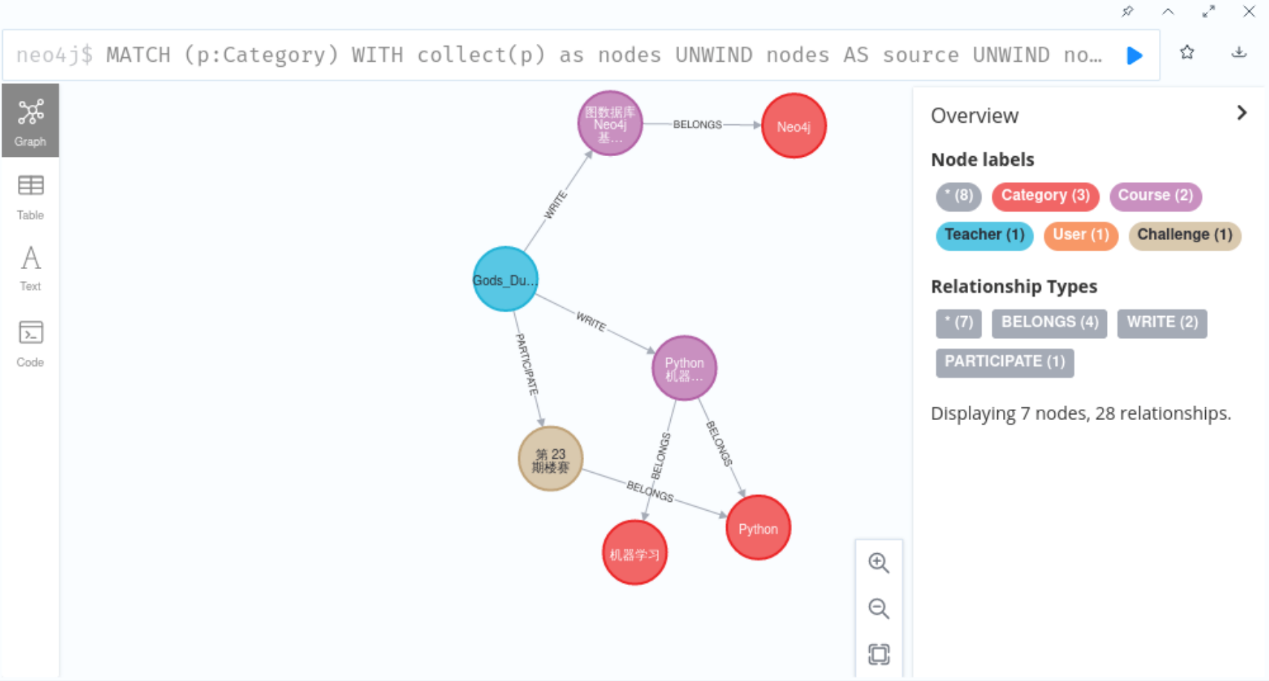
WITH paths

RETURN paths

上述语句查询了 **Category** 类型的节点之间的所有的最短路径，子句 WITH source, target WHERE id(source)<>id(target) 的目的是为了保证出入节点不相同。



在**图示图**中结果展示如下：



## 演练小结

在本演练中，给出了大量实例（吼吼~~~），练习了与图数据查询相关的关键字，数据格式，运算符和函数。在下一演练中，将演练图数据的修改语法。

# 演练 3 Cypher 图数据修改

## 演练介绍

在本演练中，将接触图数据的修改，首先演练节点关系的属性的修改，然后演练图数据的添加创建，最后学习图删除操作来了解这所有的演练。

#### 技能点

* 图属性的修改
* 图数据添加创建
* 节点标签删除
* 图删除
* 清空数据库

## 环境配置

**若是从前面的演练中进入，因 Neo4j 相关配置已保存，无需再次配置，可以直接跳过，继续后续的演练内容。**

#### Neo4j 数据导入与启动

（略）。

## 属性修改

在 Cypher 中使用 SET 进行节点标签、节点和关系属性的修改。

基本语法为 MATCH $节点或关系 SET $标签或属性。

#### 标签修改

MATCH (n {name: 'Gods\_Dusk'})

OPTIONAL MATCH (m {name: 'Neo4j'})

SET n:Male, m:Database:NoSQL

RETURN n.name, labels(n), m.name, labels(m)

使用 SET 对标签进行修改时，不会覆盖原本存在的标签，上述语句分别对 **Gods\_Dusk** 节点和 **Neo4j** 节点设置了 Male 和 Database，NoSQL 标签。



#### 属性修改

节点和关系的属性的修改包括属性的添加，替换，移除，属性值的修改。

属性添加使用运算符 += 对节点进行属性更新。

MATCH (n:Category {name: 'Python'})

SET n += {languageType: 'interpreted'}

RETURN properties(n)

上述语句对 **Python** 节点添加了属性 **languageType**，值为 **interpreted**。



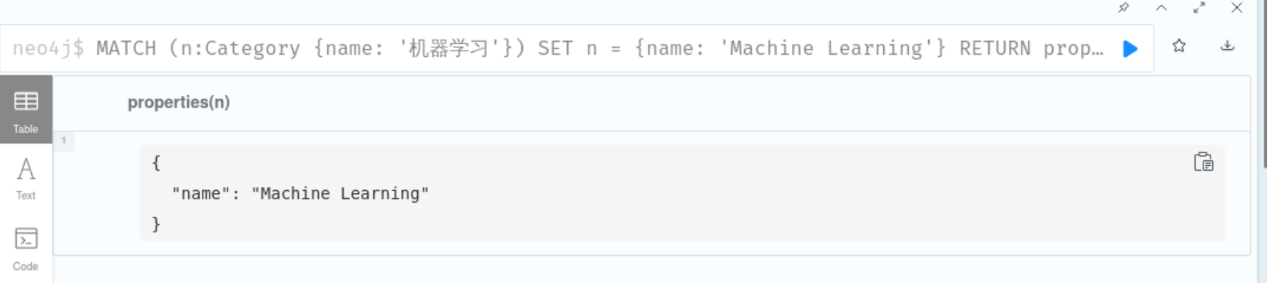
属性替换使用运算符 = 对节点进行属性替换。

MATCH (n:Category {name: '机器学习'})

SET n = {name: 'Machine Learning'}

RETURN properties(n)

上述语句对 **机器学习** 节点的属性进行了替换。



属性值的修改也使用运算符 =，与属性替换不同的是 = 作用于属性的 key 上，同时 key 不存在的话就会创建。

MATCH (n:Category {name: 'Neo4j'}) WITH properties(n) AS pn, n AS n

SET n.name = '图数据库 Neo4j', n.type = 'NoSQL'

RETURN pn, properties(n)

上述语句对 **Neo4j** 节点的属性进行了修改，其中 **name** 属性修改前存在，被修改后变成 **图数据库 Neo4j**。**type** 属性在被赋值前不存在，在这里直接被赋值为 **NoSQL**。



同样， = 也可以用于属性的移除，若要移除一个属性值，直接对其赋值为 null，移除属性则直接赋值为 {}。

## 图数据添加

在 Cypher 语法中图数据的添加有两种方式，直接添加 CREATE 与合并添加 MERGE。

#### CREATE 语句

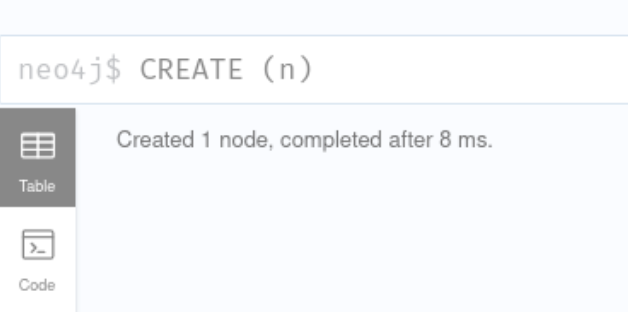
关键字 CREATE 用于创建节点，关系和路径。CREATE 基于图结构的创建的语法为：CREATE $图结构，RETURN 为可选项，当不使用 RETURN 时，**图示图**仅显示创建过程中的信息。

#### 节点创建

在 Cypher 编辑器中输入

CREATE (n)

后创建了一个无标签无属性的节点。



同理可以创建多个带标签和属性的节点。

CREATE (n:Medal {name: '中国人民的老朋友', medalId: 25}), (m:Medal {name: '年度优秀作者', medalId: 29}), (k:User:Teacher {name: '钟鼓楼', userId: 8504})

RETURN n.name, m.name, k.name

上述语句创建了 3 个节点，前两个标签为 **Medal**，后一个标签为 **User** 和 **Teacher**。



##### 关系创建

在关系创建时，需要先找到关系对应的两个节点，然后再在这两个节点之间创建关系。

MATCH (a:User {name: 'Gods\_Dusk'})

OPTIONAL MATCH (b:Medal {medalId: 29})

CREATE (a)-[r:GET\_MEDAL {description: a.name + ' get medal ' + b.name}]->(b)

RETURN type(r), r.description



#### 路径创建

在 Cypher 编辑器中输入：

CREATE p=(:User {name: '石头山', userId: 13})-

[:WRITE]->

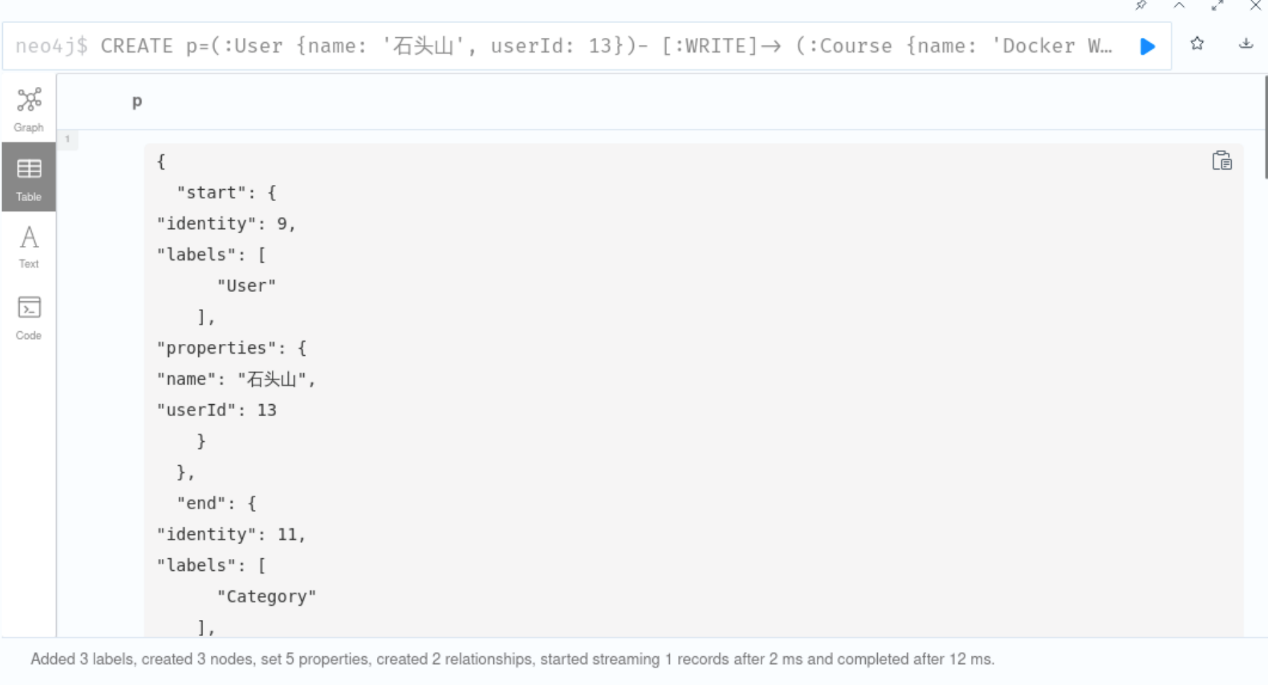
(:Course {name: 'Docker Workshop - 5小时入门 Docker', courseId: 738})-

[:BELONGS]->

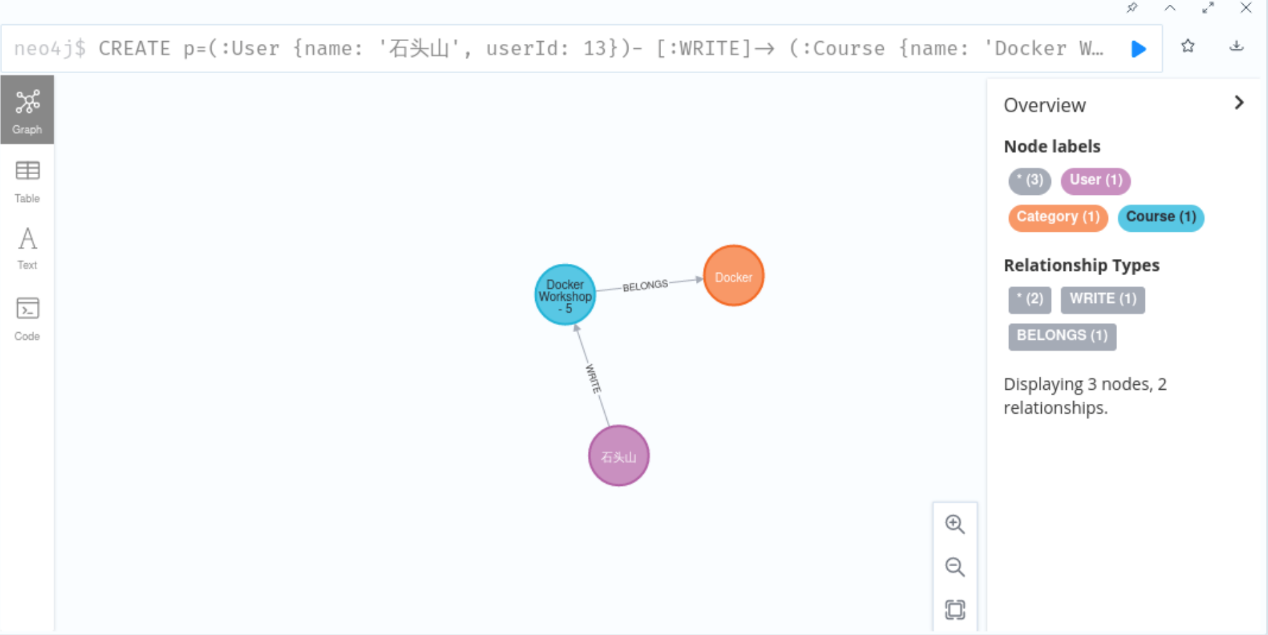
(:Category {name: 'Docker'})

RETURN p

创建并返回了一个包含 3 个节点，2 个关系的路径。这里要注意的是，创建路径时必须注明关系的方向。



在**图示图**中结果展示如下：



### MERGE 语句

MERGE 命令用于在图中搜索给定图模式，如果模式存在，则返回结果，如果不存在于图中，则进行创建。 如 CREATE 语句：

CREATE (n:TestCreate {name: 'test create'})

RETURN n.name, id(n)

执行若干次，则可以看到在侧边栏的数据库信息中，节点数量一直在增加，同时返回的内置 id 也是不同的。

而语句

MERGE (n:TestMerge {name: 'test merge'})

RETURN n.name, id(n)

执行若干次，节点只增加了一个。

从而可以认为 MERGE = MATCH + CREATE，因此，在对图数据进行添加时，若想要跳过已存在的节点或关系，使用 MERGE 命令，若不关心重复节点或关系，则使用 CREATE 命令。

同时，MERGE 后可以跟 ON MATCH 和 ON CREATE 子句来区分是否存在节点来进行后续操作。

MERGE (n:TestMerge {name: 'test'})

ON MATCH SET n.firstTest = false

ON CREATE SET n.firstTest = true

RETURN n.name, n.firstTest

上述语句第一次运行时创建了标签为 **TestMerge** 的 **test** 节点，因为节点之前不存在，执行了 ON CREATE 子句，将 **firstTest** 属性设置为 true，之后再运行则会执行 ON MATCH 子句，将 **firstTest** 属性设置为 false。

## 图数据删除

图数据的删除包括节点标签删除，属性删除，节点，关系和路径的删除，图数据清空。

#### REMOVE 语句

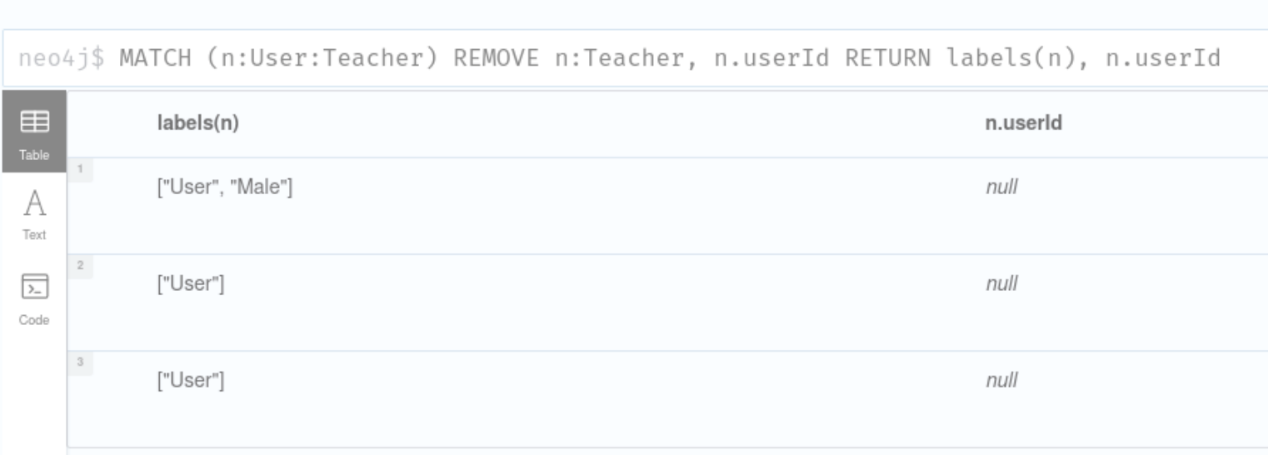
REMOVE 用于节点标签和属性值的删除，基础语法为 MATCH $图数据 REMOVE $节点标签或属性值。

MATCH (n:User:Teacher)

REMOVE n:Teacher, n.userId

RETURN labels(n), n.userId

上述语句先查询了标签为 **User**，**Teacher** 的节点，然后删除了 **Teacher** 标签和 **userId** 属性。



同时，REMOVE 只能删除属性值，要删除全部属性只能使用 set $节点或关系 = {} 语法。

#### DELETE 语句

DELETE 用于节点，关系和路径的删除，基础语法为 MATCH $图数据 DELETE $节点，关系或路径。

MATCH p=({name: '石头山'})-->()-->()

DELETE p

上述语句删除了以 **石头山** 节点为起点，长度为 2 的路径。



这里要注意的是，因为不存在孤立的关系，所以若要删除一个带关系的节点，需要同时删除该节点所有的关系。

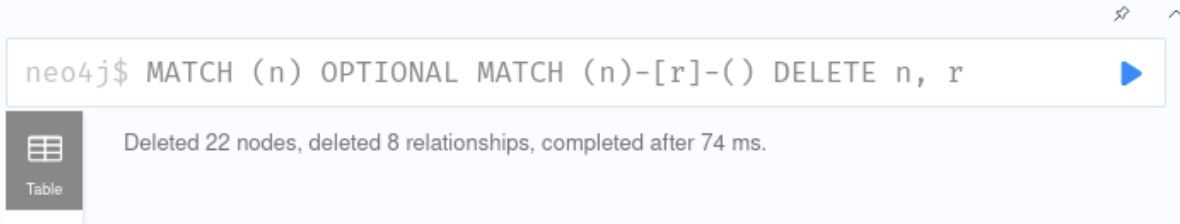
即，若要删除路径(a)-[d]-(b)-[e]-(c) 中的 a，b 节点，则需要同时删除关系 d，e。

因此，若要清空数据库，即删除所有的节点和关系，可以先使用 MATCH 找到所有的节点，再使用 OPTIONAL MATCH 查询节点是否存在关系，最后将其全部删除。

MATCH (n)

OPTIONAL MATCH (n)-[r]-()

DELETE n, r



此外，Cypher 还提供了 DETACH 关键字，配合 DELETE 进行数据库的清空。

慎用！

MATCH (n)

DETACH DELETE n

## 上述命令，相当于清空 neo4j 数据库！

## 演练总结

通过以上 3 个演练，大概涉及 Neo4j 所有的基础知识，基于此，可以很方便的构建图数据库，并对其进行数据管理。

以上！