第5章 图数据库

在关系数据库中，数据及其联系被分散存储到许多基本表中，这导致难以发现数据之间的关联关系，也难以进行知识发现和知识推理。为了对现实世界中事物间的关系进行建模，图数据库应运而生，它被越来越多地用来描述体量巨大的实体及其联系，尤其是近年来知识图谱和属性网络的研究，图数据库被用来存储实体及其关系，为知识推理提供了强有力的支持。Neo4j 是当前较流行的图数据库，支持事务特性和分布式集群架构，并提供了功能强大的查询语言和图分析算法。本章将主要介绍图基本概念、图数据模型、Neo4j 概述和 Neo4j 查询语言等内容。

5.1 图的基本概念

图是一种数据结构，能够对一组实体及其关系建模，其中实体是图中的节点，关系是图中的边。本节将围绕这两个基本概念对图的相关概念进行介绍，这些内容对于理解图数据库非常重要。

5.1.1 节点

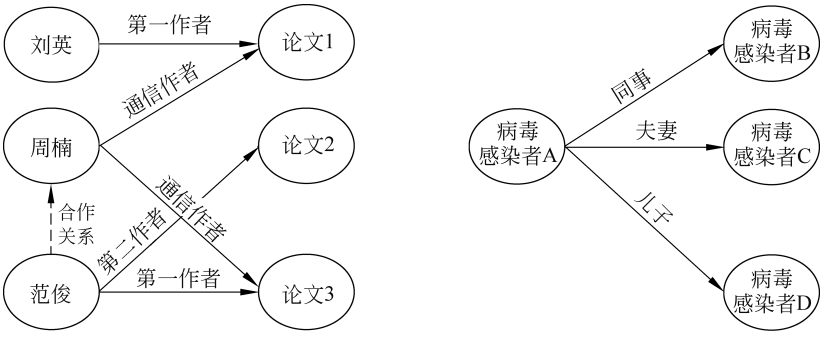
节点(Node)也称顶点(Vertex)，是图的基本元素之一。节点的总数量称为图的阶。现实世界能够相互区别的事物都可以表示为图的节点，如作者、论文、客户、商品、公司和城市等。例如，图 5-1(a)所示的论文发表关系图包括作者和论文两类节点，每个作者节点代表一个实体，每个论文节点也代表一个实体，且这些实体可以相互区别。图 5-1(b)所示的病毒传播关系图，每个感染者都是一个实体。

节点一般具有一个或多个标签，用来表示节点的类型，图 5-1(a)的节点“刘英”的标签是“作者”，节点“论文 1”的标签是“论文”。节点也可以有一组属性，作者节点可以有姓名、性别、职称、工作单位、研究领域等属性。

5.1.2 边

边(Edge)是实体之间的关系(联系)。与某节点相连的边的数量称为该节点的度数，图中边的总数量称为图的尺寸。在图 5-1(a)中，作者“刘英”发表了“论文 1”，那么在节点“刘英”和节点“论文 1”之间就存在一条边。通过“边”能够直观地表达实体之间的关系。

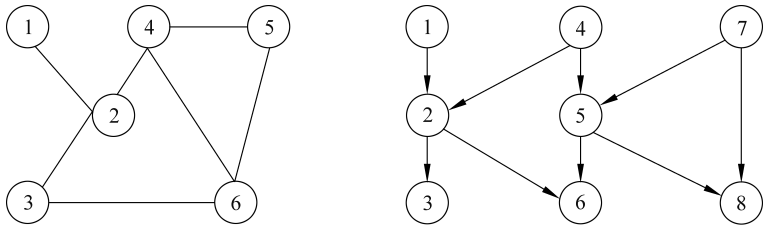
边可以有类型、属性、权重、方向等其他辅助信息。首先，边是有类型的，如发表关系、购买关系、合作关系等，代表关系的类型；其次，边也可以有属性，如发表关系中作者排序、是否



(a) 论文发表关系图　　(b) 病毒传播关系图

图 5-1　两个关系图示例

通信作者等都可以作为边的属性；再次，边也可以有权重，权重是关系重要性的度量，如距离、成本、时间等；最后，边还可以有方向，如果所有边都有方向，则称为有向图，如果所有边都没有方向，则称为无向图。在有向图中，边的起始节点称为头节点，结束节点称为尾节点。图 5-2(a)是一个无向图，而图 5-2(b)是一个有向图。



(a) 无向图　　(b) 有向图

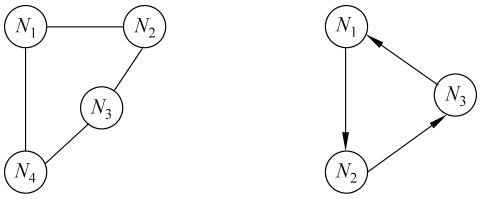
图 5-2　无向图和有向图

现实中，有些关系是以显式方式存在的，而有些关系却是隐含的。比如两个作者共同发表了一篇论文，那么这两个作者和这篇论文都有一条边。根据这两个边，通过知识推理可以得知这两个作者是合作关系，也就是这两个作者之间也存在一条边。基于已有关系预测可能存在关系的过程称为“链接预测”，反映在图上就是节点间关系的“补全”，这是图的一个重要应用。

5.1.3　路径

路径(Path)是从一个节点(开始节点)到另一个节点(结束节点)途径的所有节点组成的序列(包含开始节点和结束节点)。路径长度是指序列中边的个数。如果路径中第一个节点和最后一个节点相同，那么此路径称为“回路”(或“环”)。如果路径中各节点都不重复，那么此路径又被称为“简单路径”；同样，除第一个节点和最后一个节点，若回路中的其他节点互不重复，则此回路被称为“简单回路”(或“简单环”)。

如果图中从一个节点到另一个节点至少存在一条路径，则称这两个顶点是连通的。例如图 5-3(a)中，虽然节点 N_1 和 N_3 没有直接关联，但从 N_1 到 N_3 存在两条路径，分别是 $N_1-N_2-N_3$ 和 $N_1-N_4-N_3$，因此称 N_1 和 N_3 之间是连通的。



(a) 连通图　　(b) 强连通图

图 5-3　连通图与强连通图

在无向图中，如果任意两个顶点之间都存在一条路径，则称该无向图为连通图。在有向图中，对任意两个节点 N_i 和 N_j，满足从 N_i 到 N_j 以及从 N_j 到 N_i 都连通，也就是都含有至少一条通路，则称该有向图为强连通图。图 5-3(b)就是一个强连通图，任意两个节点之间都存在一个通路。

5.1.4　遍历

图的遍历(Traversal)是从指定的节点出发按照某种策略沿着边访问所有节点，使每个节点仅被访问一次，这个过程称为图的遍历。遍历过程中得到的节点序列称为遍历序列。根据遍历策略方法的不同，分为深度优先搜索(DFS)和广度优先搜索(BFS)。

深度优先搜索是从某个节点出发，首先访问该节点，然后依次从它的各个未被访问的邻接点出发深度优先搜索遍历图，直至图中所有和该节点有路径相通的节点都被访问到；若此时尚有其他节点未被访问到，则另选一个未被访问的节点作起始点，重复上述过程，直至图中所有节点都被访问到为止。图 5-2(a)按深度优先策略遍历的一个序列是：1-2-3-6-5-4。

广度优先搜索是从图中某节点出发，在访问了该节点之后，依次访问该节点的所有邻接点，然后分别从这些邻接点出发依次访问它们的邻接点，并使得先被访问的节点的邻接点先于后被访问的节点的邻接点被访问，直至图中所有已被访问的节点的邻接点都被访问到；如果此时图中尚有节点未被访问，则需要另选一个未曾被访问过的节点作为新的起始点，重复上述过程，直至图中所有节点都被访问到为止。图 5-2(a)按宽度优先策略遍历的一个序列是：1-2-3-4-6-5。

图数据库(如 Neo4j)提供了一套高效的遍历算法(API)，可以指定遍历规则，然后自动按照遍历规则进行遍历，并返回遍历结果。

5.2　图数据模型

图数据库(Graph Database)是基于图论开发的一种 NoSQL 数据库，无论是数据存储，还是数据操作，都以图论为基础，通过直观的方式建模客观事物之间的复杂关系。图中的基本元素包括节点和边，分别对应现实世界中的实体和关系(也称为联系)。图数据库的基本任务是对图中的节点和关系进行创建、读取、更新、删除、查询和分析等。

图数据模型定义了图中节点和关系的建模、存储和实现方法。常用的图数据模型包括三种：属性图模型、三元组模型和超图模型。

5.2.1 属性图模型

属性图模型是一种常用且比较直观的图数据模型，本章介绍的 Neo4j 图数据库就采用了该模型。属性图的主要特点如下。

(1) 属性图由节点和关系(联系)组成。

(2) 节点可以有多个属性和属性值(键值对)。

(3) 节点可以有一个或多个标签。

(4) 关系只有一个类型，并总是从一个开始节点指向一个结束节点。

(5) 关系也可以有多个属性和属性值(键值对)。

属性图能够比较方便地存储图数据。图 5-4 是由电影、演员、导演等实体及其关系构成的一个图，节点就是各类实体，边就是实体间的关系。

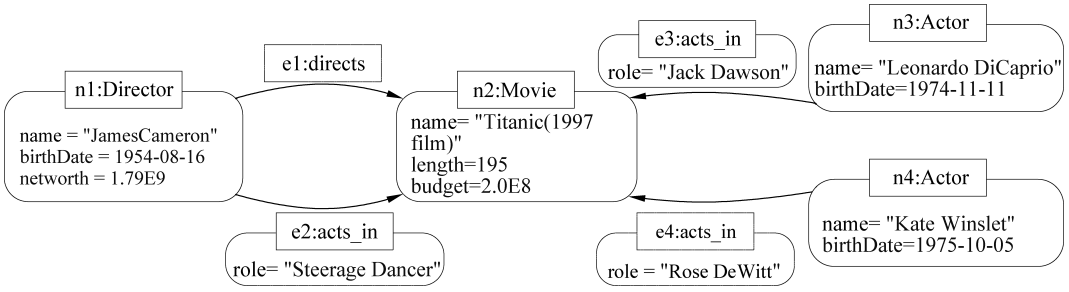


图 5-4 电影关系属性图模型

采用属性图模型，可以通过以下方式对图 5-4 所示的属性图进行存储。

(1) 节点集合

节点 $N=\{n1, n2, n3, n4\}$。

(2) 关系集合

关系 $E=\{e1, e2, e3, e4\}$，其中 $e1=(n1, n2)$，$e2=(n1, n2)$，$e3=(n3, n2)$，$e4=(n4, n2)$。

(3) 节点标签集合

Label(n1)=Director，Label(n2)=Movie，Label(n3)=Actor，Label(n4)=Actor。

(4) 关系类型集合

Type(e1)=directs，Type(e2)=acts_in，Type(e3)=acts_in，Type(e4)=acts_in。

(5) 属性集合

Attr(n1，name)='James Cameron'，　Attr(n1，birthDate)=1954-08-16，

Attr(n1，networth)=1.79E9，　Attr(n2，name)=' Titanic(1997 film)'，

Attr(n2，length)=195，　Attr(n2，budget)=2.0E8

Attr(n3，name)= 'Leonardo DiCaprio'，　Attr(n3，birthDate)=1974-11-11

Attr(n4，name)= ' Kate Winslet'，　Attr(n3，birthDate)=1975-10-05

Attr(e2，role)='Steerage Dancer'，　Attr(e3，role)='Jack Dawson'，

Attr(e4，role)=' Rose DeWitt'。

5.2.2 三元组模型

三元组模型是另外一种重要的图数据模型，包含主谓宾的数据结构。通过三元组表达实体与实体之间的关系，或者属性与属性值之间的关系，有以下两种形式。

- （实体，关系，实体）：该三元组表达了实体与实体的关系，分别称为头实体和尾实体；
- （实体，属性，属性值）：该三元组表达了实体与属性之间的关系。

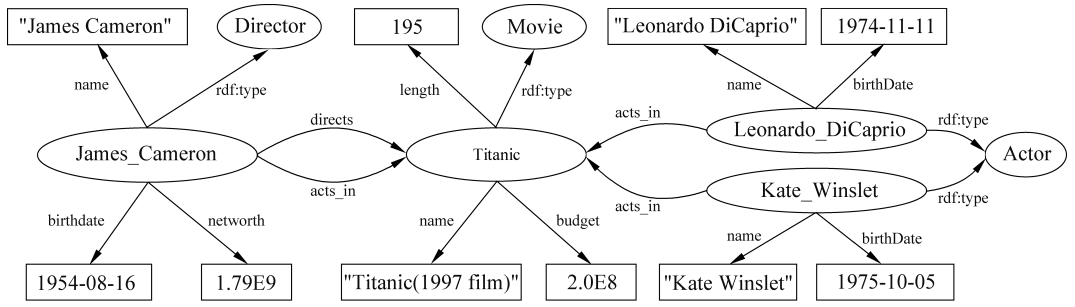


图 5-5 电影关系三元组模型

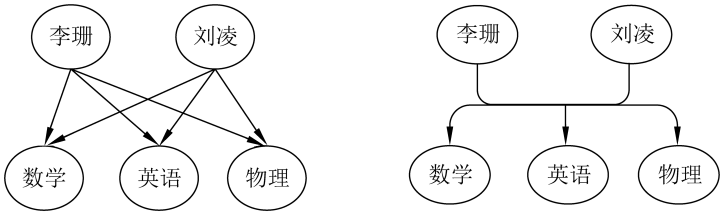
图 5-4 的属性图模型可以转换为图 5-5 所示的三元组模型，按以下三元组方式存储。

(1) (James_Cameron, rdf:type, Director),
(2) (James_Cameron, name, James Cameron),
(3) (James_Cameron, birthDate, 1954-08-16),
(4) (James_Cameron, networth, 1.79E9),
(5) (James_Cameron, directs, Titanic),
(6) (James_Cameron, acts_in, Titanic),
(7) (Titanic, rdf:type, Movie),
(8) (Titanic, length ,195),
(9) (Titanic, name, Titanic(1997 film)),
(10) (Titanic,budget ,2.0E8),
(11) (Leonardo_DiCaprio,name, Leonardo DiCaprio),
(12) (Leonardo_DiCaprio,birthDate, 1974-11-11),
(13) (Leonardo_DiCaprio,rdf:type, Actor),
(14) (Leonardo_DiCaprio,acts_in, Titanic),
(15) (Kate_Winslet,name, Kate Winslet),
(16) (Kate_Winslet,birthDate, 1975-10-05),
(17) (Kate_Winslet,rdf:type, Actor),
(18) (Kate_Winslet,acts_in, Titanic)。

5.2.3 超图模型

超图是一种广义的图数据模型，它允许一条边关联任意数量的节点，无论是开始节点还

是结束节点，这样的边称为超边。相比于属性图，超图更适合对多对多关系进行建模，通过超边进行关联。如图 5-6(a)所示，通过属性图对学生和课程的多对多选修关系进行建模，每个学生可以选修多门课，同样每门课程可以有多个学生选修。图 5-6(b)通过超图对多对多的选修关系进行建模，共有 1 个超边，该超边的开始节点有两个，结束节点有三个。



(a) 属性图　　(b) 超图

图 5-6　属性图和超图的比较

5.3　Neo4j 概述

5.3.1　特点

目前主流的图数据库有 Neo4j，FlockDB，GraphDB，InfiniteGraph，Titan，JanusGraph，Pregel 等，这些图数据库主要采用属性图数据模型。Neo4j 是一个基于 Java 语言开发的图数据库，适用于大量复杂、互连接、低结构化的数据，避免了由于连接操作产生的性能问题，其特点如下。

(1) 高性能。与关系数据库的关系查询相比，图数据库直接将所有的关系放在实体的列表中，避免了在关系数据库中频繁地通过连接操作进行关系查询的问题，提高了查询性能。

(2) 高可用性。Neo4j 企业版可以部署在集群中，在硬件设备损坏的情况下使数据库具备完善的数据可读写能力，具有比单台数据库处理更多的负载处理能力。

(3) **ACID** 事务特性。能够通过事务的提交和回复确保数据提交的正确性和一致性，此外，也可以通过日志对数据进行恢复。

(4) 图分析算法库。这些算法帮助用户分析图数据中的隐藏模式和结构，用户不必关心这些算法的实现细节，典型的算法如广度优先搜索算法、深度优先搜索算法、最小权重生成树(MWST)算法等。

(5) **REST** 服务接口。Neo4j 图数据库除了提供基于 Java 的客户端驱动包外，同时也支持 REST 服务访问它，使得任何支持 HTTP 访问的编程语言都可以使用 REST 服务访问图数据库。

5.3.2　免索引邻接

Neo4j 采用免索引邻接技术提高图数据的操作性能。假设在一个关系数据库中设计了三个基本表，分别用来存储作者信息、发表信息和论文信息，如图 5-7 所示。在发表信息表中，列 Rno 和 Pno 分别是外键。那么，在查询某个学者发表了哪些论文时，需要通过两个外

键将这些基本表连接起来。连接操作(Join)将这些数据读入内存,一旦涉及的数据量较大,I/O 访问延迟就会成为一个关键瓶颈。为了提高性能,通常在外键上建立索引,通过建立索引进行查询的时间复杂度为 $O(\log(n))$,这表示随着作者数量和论文数量的增加,查询时间也随之增加。

作者信息

Rno	Rname
R01	刘英
R02	周楠
R03	范俊
...	...

发表信息

Rno	Pno
R01	P01
R02	P01
R02	P03
R03	P02
R03	P03
...	...

论文信息

Pno	Ptitle
P01	AI
P02	Big data
P03	Database
P04	NoSQL
...	...

图 5-7 作者与论文之间的发表关系

图数据库通过边建立节点间的多对多联系,不需要通过连接操作进行数据查询。免索引邻接技术为每个节点维护了一个与该节点直接关联的节点、关系和属性双向链表,这样,根据节点即可快速地查找其关联的节点、关系和属性,提高了查询性能,时间复杂度降低为 $O(1)$。免索引邻接技术的优势是查询性能不会受节点和关系数量影响,每次遍历仅与该节点所涉及的数据集有关,不会随着节点数的增加而增加。

5.3.3 存储结构

Neo4j 图数据库主要有三种重要的对象,包括节点、关系和数学,下面介绍它们的存储结构。

1. 节点存储结构

在 Neo4j 图数据库中,用来存储节点的文件是 neostore.nodestore.db,该文件以记录形式保存了所有节点,每个节点记录的长度大小固定,均为 9B,格式为

```
Node:inUse+nextRelId+nextPropId
```

- inUse:若该项的值是 1,则表示该节点被正常使用;若该项的值是 0,则表示该节点被删除,占 1B。
- nextRelId:用来存储该节点的下一个关系 ID,整数,占 4B。
- nextPropId:用来存储该节点的下一个属性 ID,整数,占 4B。

节点的物理存储结构如图 5-8 所示。

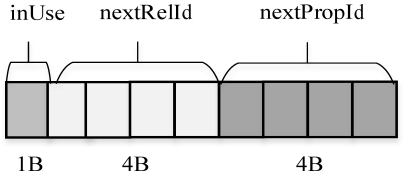


图 5-8 节点的物理存储结构

2. 关系存储结构

用来存储关系的文件名是 neostore.relationshipstore.db,该文件也以记录形式保存了所有关系,每个关系记录的长度大小固定,均为 33B,格式为

```
Relationship: inUse+firstNode+secondNode+relType+firstPrevRelId
                +firstNextrelId+secondPrevRelId +secondNextRelId+nextPropId
```

- inUse：若该项的值是 1，则表示该关系被正常使用；若该项的值是 0，则表示该关系被删除，占 1B。
- firstNode：该关系的开始节点，整数，占 4B。
- secondNode：该关系的结束节点，整数，占 4B。
- relType：该关系的类型，整数，占 4B。
- firstPrevRelId：开始节点的前一个关系 ID，整数，占 4B。
- firstNextRelId：开始节点的后一个关系 ID，整数，占 4B。
- secondPrevRelId：结束节点的前一个关系 ID，整数，占 4B。
- secondNextRelId：结束节点的后一个关系 ID，整数，占 4B。
- nextPropId：关系的最近属性 ID，整数，占 4B。

上述前后关系是在添加的时候自然形成的。如果关系是最开始添加的，则没有前一个关系；如果关系是最后一个添加的，则没有后一个关系；其他中间添加的关系都应该有前一个关系和后一个关系；由此，这些先后添加的关系将形成一个列表。

关系的物理存储结构如图 5-9 所示。

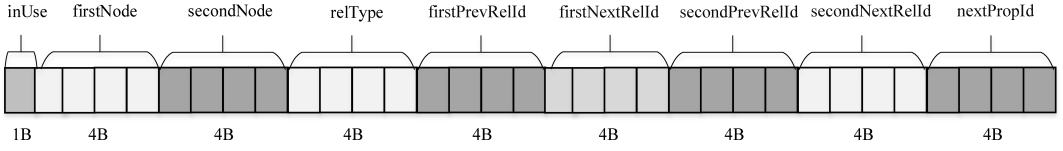


图 5-9　关系的物理结构

3. 属性存储结构

用来存储属性的文件名是 neostore.propertystore.db，该文件也以记录形式保存了所有属性，每个属性记录的长度大小固定，默认 41B，格式为

```
Property: inUse+NextPropId+PrevPropId+DEFAULT_PAYLOAD_SIZE
```

- inUse：表示属性是否可用，占 1B。
- NextPropId：下一个属性的 ID，占 4B。
- PrevPropId：前一个属性的 ID，占 4B。
- DEFAULT_PAYLOAD_SIZE：属性块，存储了属性类型和属性值，占 32B。

基于以上文件存储结构，可以快速地查找节点、关系及其属性。如果需要读取节点，只要根据节点的 ID 号（整数值）乘以节点大小（9B）就可以方便地计算出节点在存储文件中的位置。如果需要读取属性，只需要从节点的第一个属性的指针开始，遍历属性列表就可以获得；如果要查找节点的关系，只需要根据节点的第一个关系，遍历整个双向链表就可以找到关系。

图 5-10 给出了一个图存储结构实例，该图包含 5 个节点和 6 个关系，其中 5 个节点的存储记录如下所示。

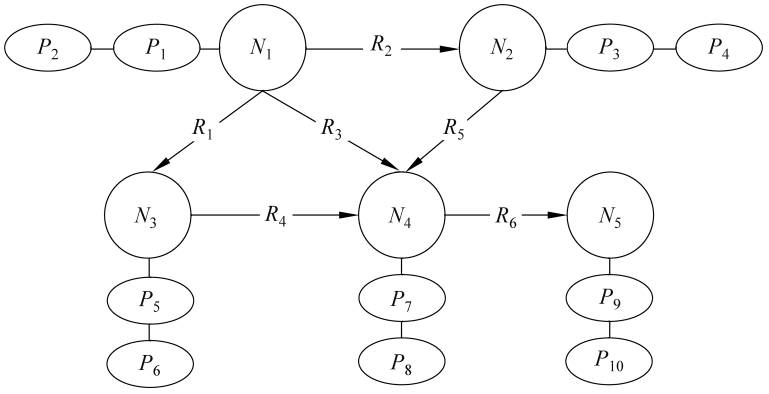


图 5-10 图存储结构实例

- Node[1, used=true, rel=1, prop=1]
- Node[2, used=true, rel=2, prop=3]
- Node[3, used=true, rel=1, prop=5]
- Node[4, used=true, rel=3, prop=7]
- Node[5, used=true, rel=6, prop=9]

6 个关系的存储记录如下所示。

- Relationship[1, used=true, source=1, target=3,type1, spre=-1,snext=2, tprev=-1,tNext=4, prop=-1]
- Relationship[2, used=true, source=1, target=2,type2, spre=-1,snext=3, tprev=-1,tNext=5, prop=-1]
- Relationship[3,used=true, source=1, target=4,type3, spre=-1,snext=-1, tprev=-1,tNext=6, prop=-1]
- Relationship[4, used=true, source=3, target=4,type4, spre=1,snext=-1, tprev=3,tNext=6, prop=-1]
- Relationship[5, used=true, source=2, target=4,type5, spre=2,snext=-1, tprev=4,tNext=6, prop=-1]
- Relationship[6, used=true, source=4, target=5,type6,spre=5,snext=-1, tprev=-1,tNext=-1, prop=-1]

5.4 Neo4j 查询语言

与关系数据库的标准查询语言 SQL 类似，Neo4j 图数据库拥有自己的查询语言 Cypher，也称为 CQL(Cypher Query Language)，该查询语言已经成为图数据库事实上的标准语言。

Cypher 也是一种声明式查询语言，允许用户不必编写图形结构的遍历代码，即可对图

数据进行高效查询和更新。为了方便在上下文中引用查询结果，Cypher 允许定义各类变量，变量的定义需要遵循以下规则。

- 关键字不区分大小写，但是用户定义的各类对象（如属性值、标签、关系类型和变量等）区分大小写。
- 变量名必须以字母开头，可以包含下画线、字母和数字。
- 变量命名规则也适用于属性的命名。
- 变量仅在一个查询块内可见，不能用于后续查询。

在本章中，关键字统一用大写表示，以区分用户自定义的各类对象。

例 5-1 查询整个图数据库的所有节点和关系，其查询语句为

```
MATCH (n) RETURN n
```

上述语句是最简单的 Cypher 语句。MATCH 语句的功能是查询指定的模式，这里的模式是一对小括号()，它表示节点模式。由于该节点模式中没有任何限定条件，因此所有节点都与之相匹配。n 是自定义的节点变量，查询的所有节点全部赋值给该节点变量。RETURN 是返回语句，即将 n 所指代的全部节点及与该节点相关的关系全部返回。图 5-11 给出了登录 Neo4j 之后的图形化界面，上述命令写入顶部的输入框中。

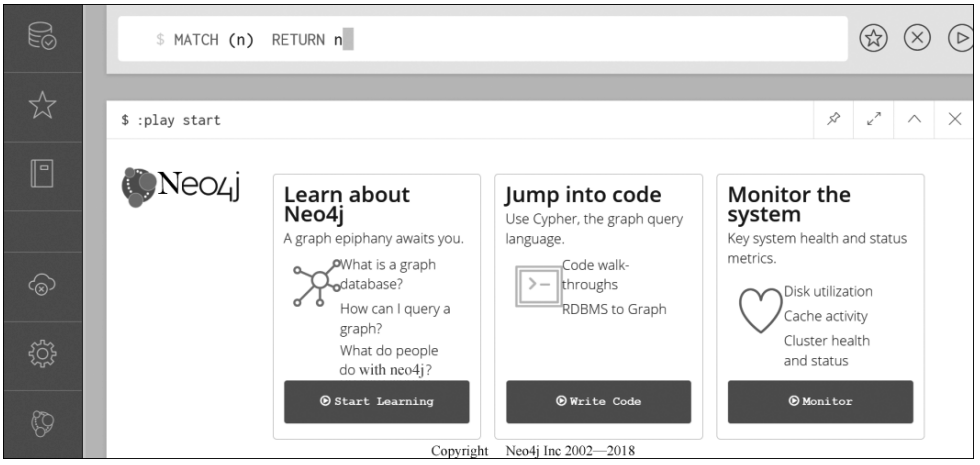


图 5-11 Neo4j 图形化界面

一般地，Cypher 语言的操作语句可以分为以下三类。

（1）写语句：这类语句的功能是实现对图数据库中各种对象的增加、删除和修改。

（2）读语句：这类语句的功能是实现对图数据库的查询和统计。

（3）通用语句：这类语句的功能是对图数据库操作施加某些限制条件。

除上述三类操作语句外，Cypher 还提供了丰富的函数，可以被用户直接调用。

5.4.1 写语句

Cypher 语言的写语句主要包括 CREATE 语句、MERGE 语句、SET 语句、DELETE 语句、REMOVE 语句、FOREACH 语句、CREATE UNIQUE 语句，用于管理节点、关系及它