

TigerGraph中文指南

首个原生并行图（NPG）系统

图数据库实战入门 —— 一个简单的电影推荐系统实现

转载自：[TigerGraph维加星](#) 作者：宋壬初 7月6日

日常生活中有各种各样的推荐系统，比如电子购物网站根据用户历史消费习惯和浏览记录进行商品推荐，以及在线视频网站根据用户历史观看记录进行视频推荐。今天，我们就来使用TigerGraph图数据库技术设计一个简单的电影推荐系统。

Table of Contents



- [1. 准备工作](#)
- [2. 创建图模型](#)
- [3. 创建数据映射发](#)
- [4. 载入数据](#)
- [5. 浏览图数据](#)
- [6. 两个电影节点之间的一条最短路径](#)
- [7. 电影推荐算法实现](#)
- [8. 总结](#)
- [9. 引用资料：](#)
- [10. 拓展阅读：](#)

准备工作

请下载并安装[TigerGraph终身免费的开发者版本\(Developer Edition\)](#)

注册用户后会收到一封包含下载链接和安装步骤的邮件。安装过程大概需要15分钟。

由于今天使用的数据集规模对于TigerGraph系统来说实在是小case，个人笔记本电脑的虚拟机就可以轻松驾驭。我们将TigerGraph系统安装在MacBook Pro搭载的虚拟机中。MacBook Pro配置为：CPU 2.5 GHz Intel Core i7, 内存16GB, 硬盘500GB Flash Storage。VMWare Fusion虚拟机配置为CPU 4 processor cores, 10GB内存, 100GB硬盘, 操作系统为xubuntu 18.04。

我们使用MovieLens 20M数据集，该数据集相关工作请参见引用资料[1]。该数据集包含了138,000位用户针对27,000部电影的2000万条评分记录。数据集的下载地址为：<https://grouplens.org/datasets/movielens/20m/>

解压缩ml-20m.zip压缩包后，我们来熟悉一下该数据集的数据格式。

movies.csv:

movieId	title	genres
1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
2	Jumanji (1995)	Adventure Children Fantasy
3	Grumpier Old Men (1995)	Comedy Romance
4	Waiting to Exhale (1995)	Comedy Drama Romance
5	Father of the Bride Part II (1995)	Comedy
6	Heat (1995)	Action Crime Thriller
7	Sabrina (1995)	Comedy Romance
8	Tom and Huck (1995)	Adventure Children
9	Sudden Death (1995)	Action

该文件总共有27,279行，除第1行是表头外，每行用3列表示一部电影，分别为电影id（movieId）、电影名称（title）和电影类型（genres）。需要注意的是该csv文件用逗号分隔不同的列，而为了处理电影名称中包含的逗号，使用双引号（”）转义title列。

ratings.csv:

userId	movieId	rating	timestamp
1	2	3.5	1112486027
1	29	3.5	1112484676
1	32	3.5	1112484819
1	47	3.5	1112484727
1	50	3.5	1112484580
1	112	3.5	1094785740
1	151	4	1094785734
1	223	4	1112485573
1	253	4	1112485573

该文件总共有20,000,264行，除第1行是表头外，每行用4列表示一位用户对一部电影的评分，分别为用户id（userId）、电影id（movieId）、评分（rating）和评分时间（timestamp）。这里的评分时间是用unix时间戳表示的。

在这个数据集中并没有提供用户的个人信息，可能是出于保护用户隐私的考虑。因此在后面的可视化展示中我们看到的用户数据都是被一个数字表示的。

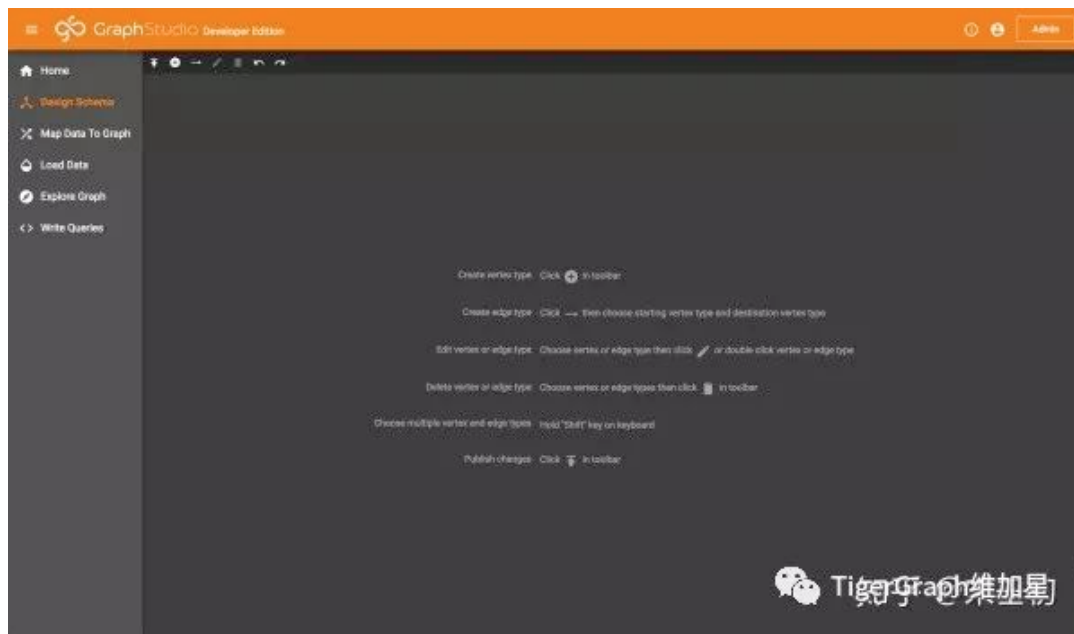
其他几个文件提供了电影标签等信息，这在我们今天的简单电影推荐算法中不会被使用。如果你有兴趣设计更加复杂的推荐算法，可以尝试将它们引入到推荐模型中获得更好的推荐效果。

创建图模型

图模型由若干节点类型（**vertex type**）和若干边类型（**edge type**）组成。可以指定边类型的源节点类型（**source vertex type**）和目标节点类型（**target vertex type**）。图模型是对现实世界的问题的一种直观的抽象。

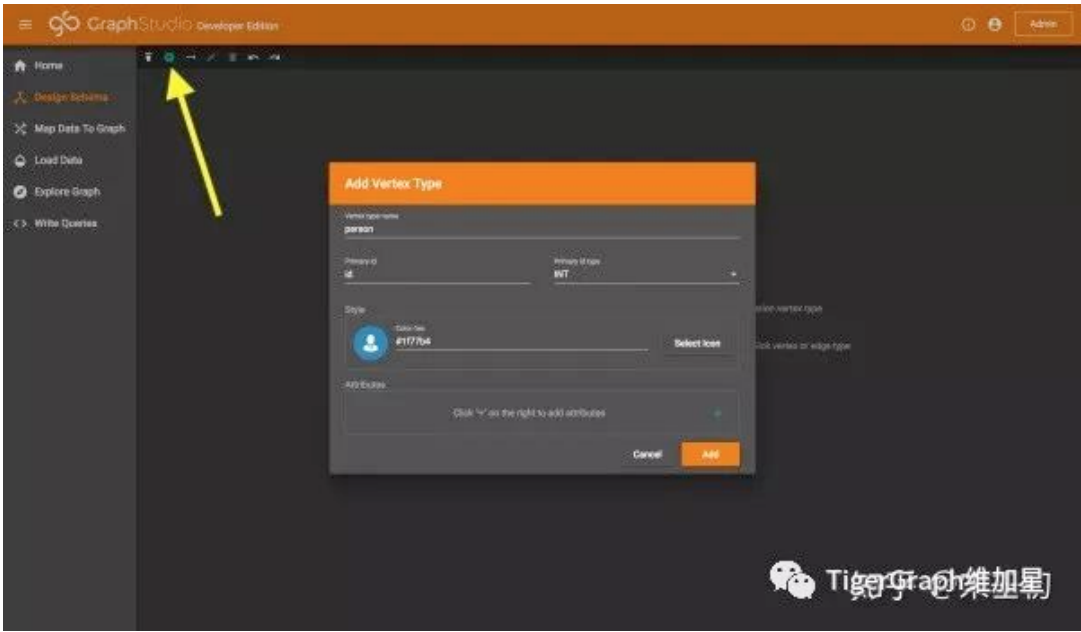
我们很容易建立电影推荐问题的模型，模型中有两种节点类型：人（**person**）和电影（**movie**），以及一种边类型：打分（**rate**）。**rate**的源节点类型为**person**，目标节点类型为**movie**。

我们使用GUI集成开发工具GraphStudio创建图模型。打开浏览器，在地址栏输入安装TigerGraph机器的IP+14240端口访问GraphStudio，载入完成后点击左侧导航栏的Design Schema项进入创建图模型页面：

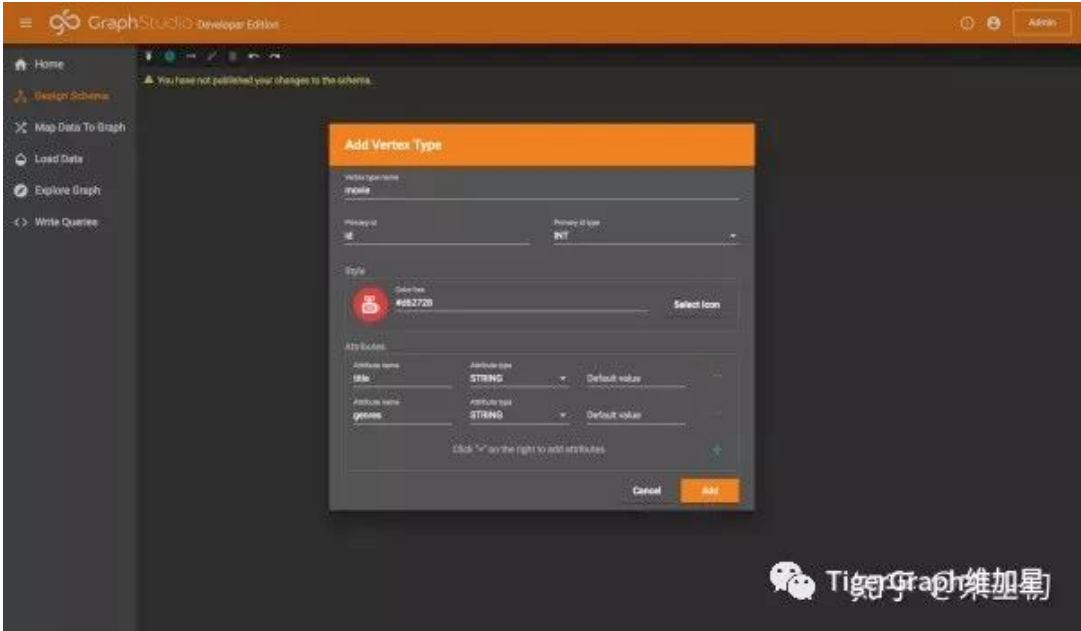


创建图模型（Design Schema）页面

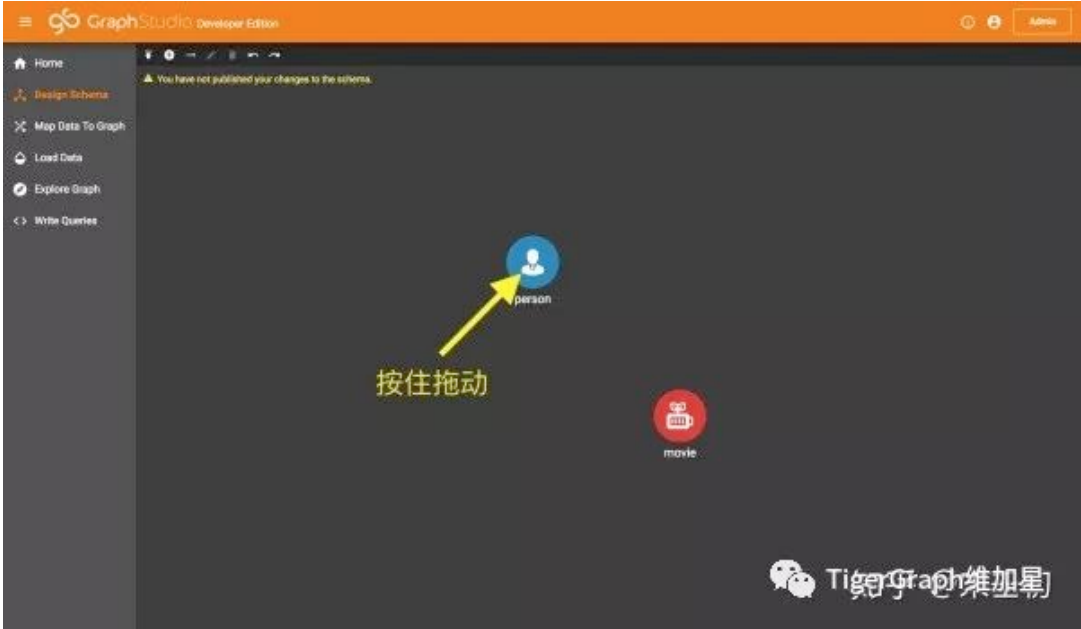
单击下图中黄色箭头所指的工具栏中的按钮即可添加节点类型，在弹出的窗口中设置节点类型名称、主键（**primary id**）名称和类型、属性（**attribute**）名称和类型，并根据语义选择节点类型的颜色和图标。我们首先添加**person**节点类型：



添加person节点类型然后添加movie节点类型



添加Movie节点类型
添加完毕后可以通过鼠标拖动调整节点类型的位置：



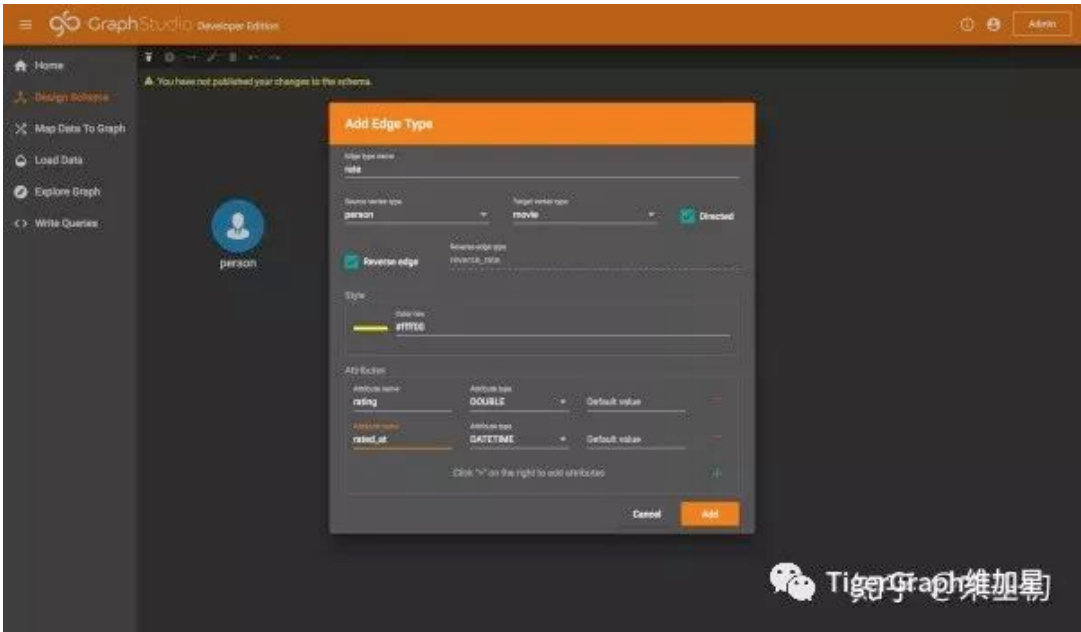
节调整节点类型的位置

单击下图第一步中黄色箭头所指的工具栏中的按钮进入添加边类型模式，然后如第二步点击源节点类型，然后如第三步点击目标节点类型。



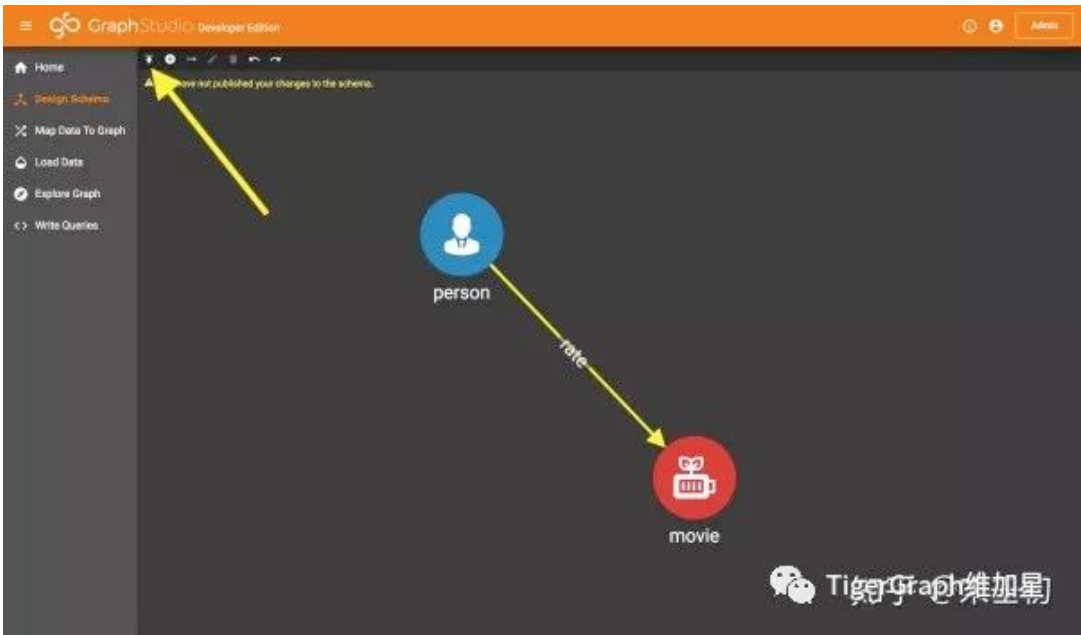
添加一种边类型

在弹出的窗口中设置边类型名称、边类型的有向性（**directed**）、属性（**attribute**）名称和类型，并可以选择边类型的颜色。我们输入**rate**边类型的信息：



输入rate边类型的信息

至此，我们完成了图模型的创建。可以用鼠标滚轮缩放图模型，也可以用鼠标按住工作面板的空白处拖动整个图模型。点击工具栏中的发布按钮将图模型发布到TigerGraph系统中。整个发布大概需要2分钟。



发布图模型

创建数据映射发

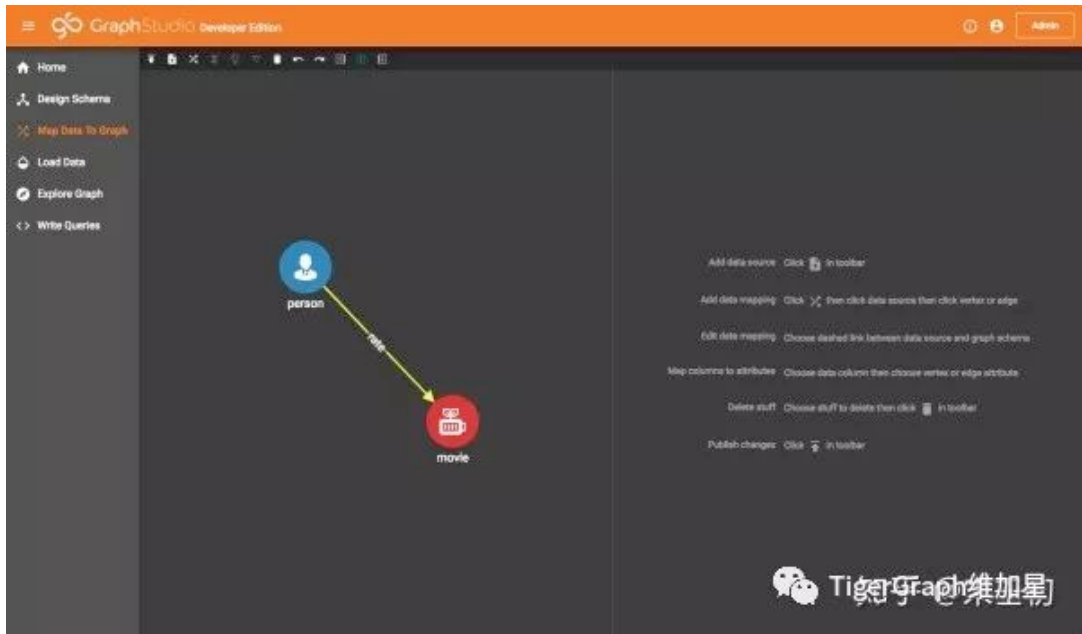
数据映射（data mapping）指建立数据模型之间的元素的对应关系。在电影推荐的这个实例中，我们需要建立从csv文件代表的数据库模型到图模型之间的对应关系。

这里需要弄清楚模型和元素之间的关系，这种关系类似于面向对象程序设计中类（class）与实例（instance）之间的关系。我们刚刚创建的图模型描述了这些类之间的关系，而我们接

下来要向图中载入的数据（元素）则是具体的每一个人、每一部电影和每条某人对某电影的打分。

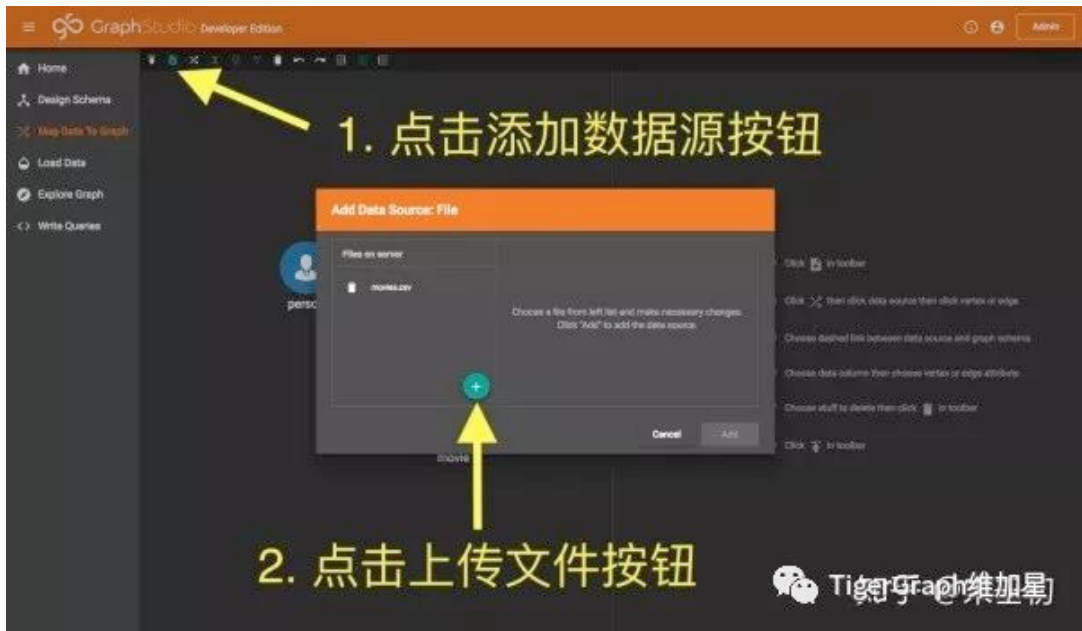
在由movies.csv文件和ratings.csv文件组成的模型中，文件表头的语义代表了该模型的结构。movies.csv文件除表头以外的每行数据代表了一个电影元素，我们需要将它映射到图模型中的电影元素。ratings.csv文件除表头以外的每行数据包含了一个（可能重复出现的）人元素和一个打分元素，我们需要将它映射到图模型的人元素和打分元素。

点击左侧导航栏的Map Data To Graph项进入创建数据映射页面：



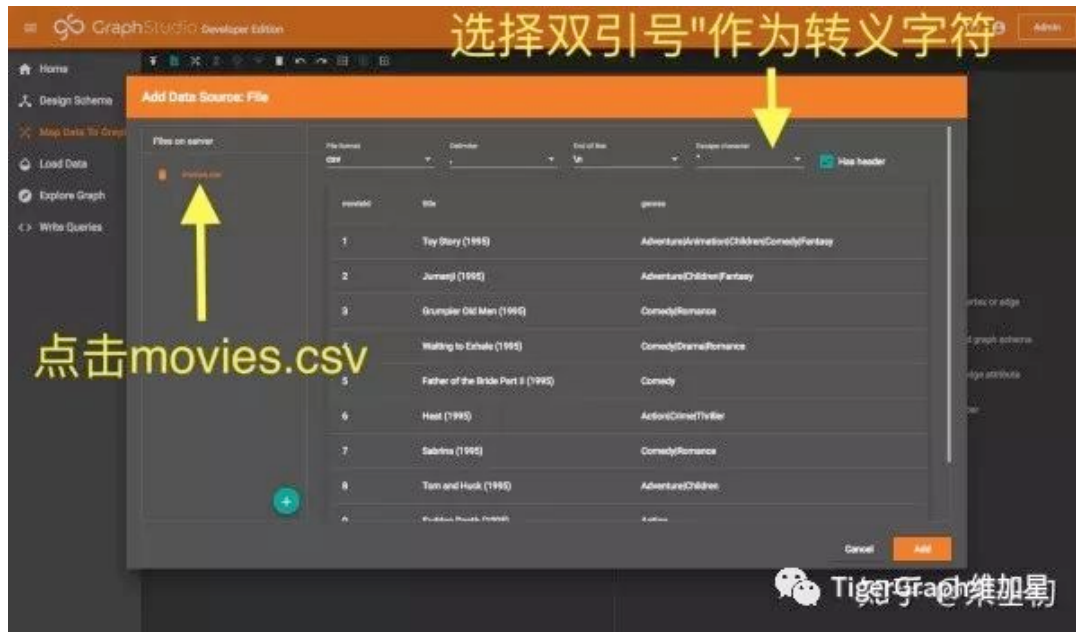
创建数据映射（Map Data To Graph）页面

我们需要将数据文件上传到TigerGraph后台。这里有两种方式：对于小于500MB的文件，可以直接通过GUI上传。点击下图中黄色箭头1所指的工具栏添加数据源按钮，在弹出的窗口中点击黄色箭头2所指的上传文件按钮，选择本机解压缩后的ml-20m数据集中的movies.csv文件上传。上传完成后在文件列表中会显示该文件：



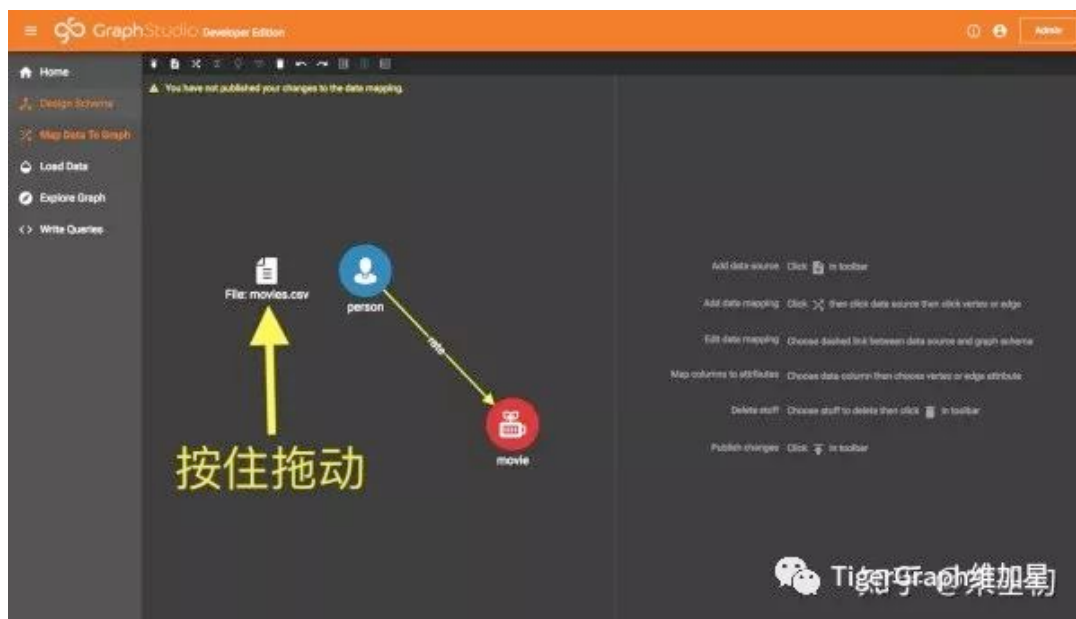
上传movies.csv

然后我们将该数据源添加到工作面板上。在Files on server列表中点击movies.csv文件，GraphStudio后台用算法智能分析数据并推断出文件的分隔符（delimiter）、换行符（end of line）和是否有表头（has header）。GraphStudio的这个推断是尽力而为的，如果不准确，用户可以自由修改这些设置，文件会马上被重新分行分列。需要注意的是GraphStudio不对转义字符（escape character）做推断。回顾前文我们提到movies.csv对于title列数据有逗号的情况采用双引号转义，所以我们需要在转义字符下拉列表中选择双引号（"）：



选择双引号作为转义字符

点击添加之后，movies.csv作为一个数据源被添加到工作面板上，表示为一个文件图标。用户可以按住这个图标拖动到任何想要的位置：



movies.csv被添加到工作面板

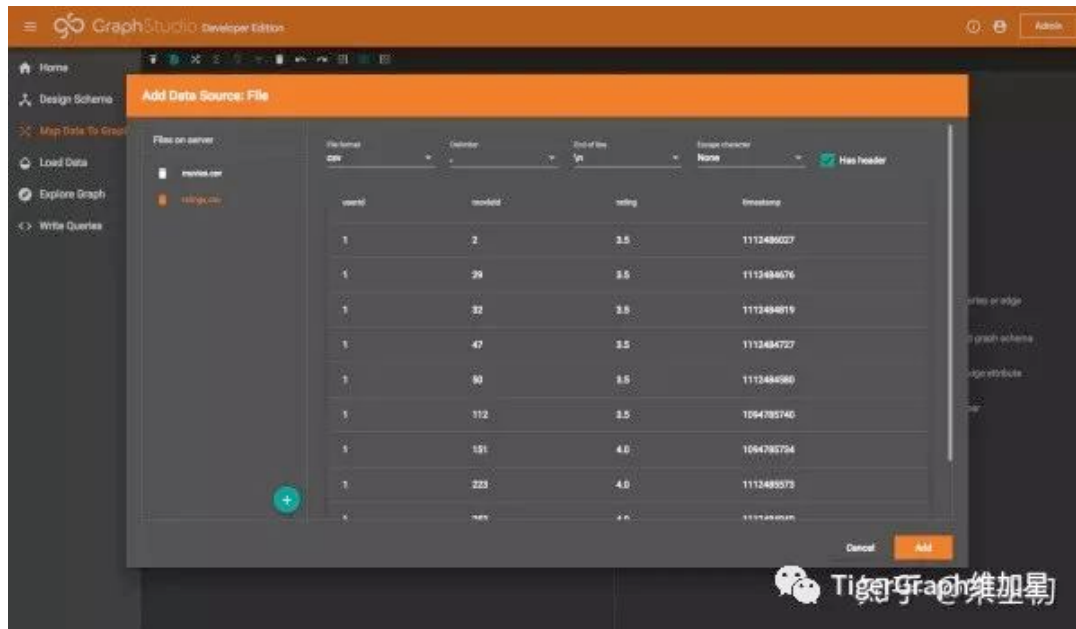
下面我们添加ratings.csv文件。出于保护系统资源的考虑，GraphStudio限制通过GUI上传单个文件不能超过500MB，而ratings.csv刚好超过了这个限制，因此我们通过scp命令（或

者任意其他方式）直接将该文件传到虚拟机的

/home/tigergraph/tigergraph/loadingData/

文件夹内（如果你在安装TigerGraph过程中没有使用默认的用户名或安装路径，则请将文件上传到相应的路径）。

再次点击工具栏添加数据源按钮，在弹出的窗口中选择ratings.csv文件添加到工作面板：



添加ratings.csv

好了，我们已经添加了所需的数据源，接下来创建数据源到图模型之间的数据映射。

首先我们将movies.csv映射到movie节点类型。点击工具栏中的映射数据到图模型按钮，然后点击数据源（movies.csv）图标，然后点击目标节点类型（movie）。这时候一条数据映射关系就被创建了：



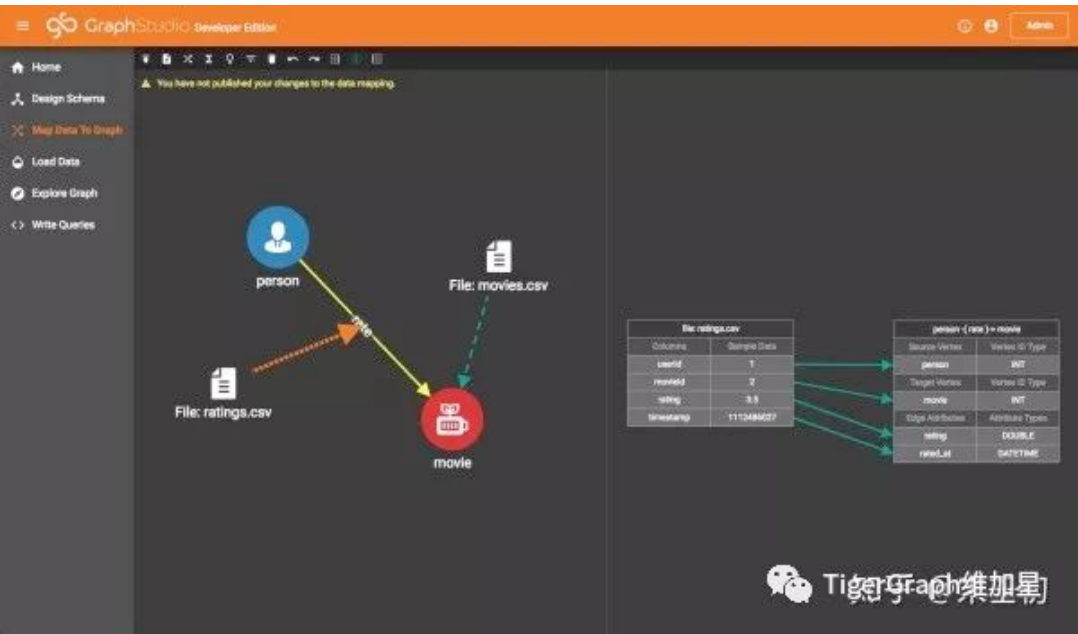
映射movies.csv到movie节点

接下来，我们需要填充映射关系的内容，即数据源中的数据如何映射到目标节点（或边）的属性。在右侧工作面板中，表示数据源movies.csv的表和表示movie节点类型的表已经静静地等在那里了。建立映射的方式非常直观，先点击数据源表中的某一行（对应于csv文件中的某一列，这种旋转90度的表达方式是ETL中普遍采用的可视化方式），再点击节点类型属性表中的某一行（对应节点的主键或某个属性），就完成了属性映射。这里我们建立了三条属性映射，你可能注意到原来显示在左侧工作面板该数据映射上面的错误信息消失了，这是因为你创建了对movie节点类型的主键的映射。对于节点来说，主键映射是必须的。而属性可以不被映射，在这种情况下当数据加载时这些未被映射的属性会使用默认值。



完成movies.csv到movie节点类型的映射

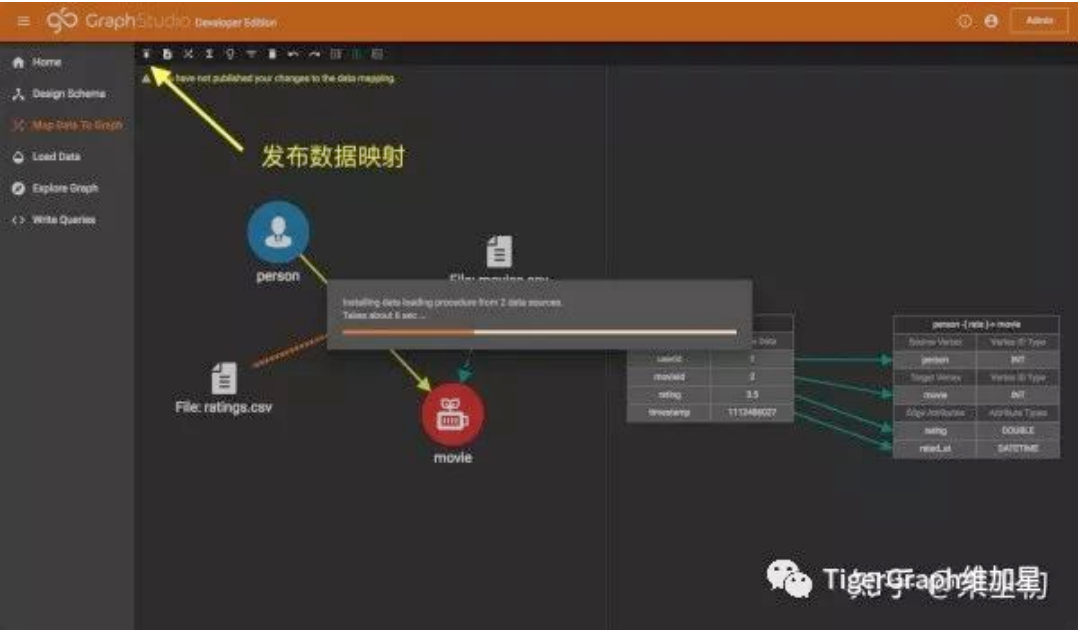
最后，我们建立ratings.csv到rate边类型的数据映射。重复与上面类似的操作，最终的映射结果为：



完成ratings.csv到rate边类型的映射

至此，我们完成了数据映射的创建。需要注意的是我们并没有映射从ratings.csv到person节点类型的映射，这是因为在ratings.csv映射到rate边类型的时候我们建立了userId列到rate边的源节点类型person的属性映射，这会在后续的数据加载中自动创建以ratings.csv中该列数据值为主键的person类型的节点。在TigerGraph系统中所有加载到相同类型的具有相同主键的节点会被合并为一个节点，默认的合并规则为属性覆盖。

最后，点击左上角的发布按钮将数据映射发布到TigerGraph系统。发布所需时间和数据映射的个数相关，这里大概需要6秒：



发布数据映射

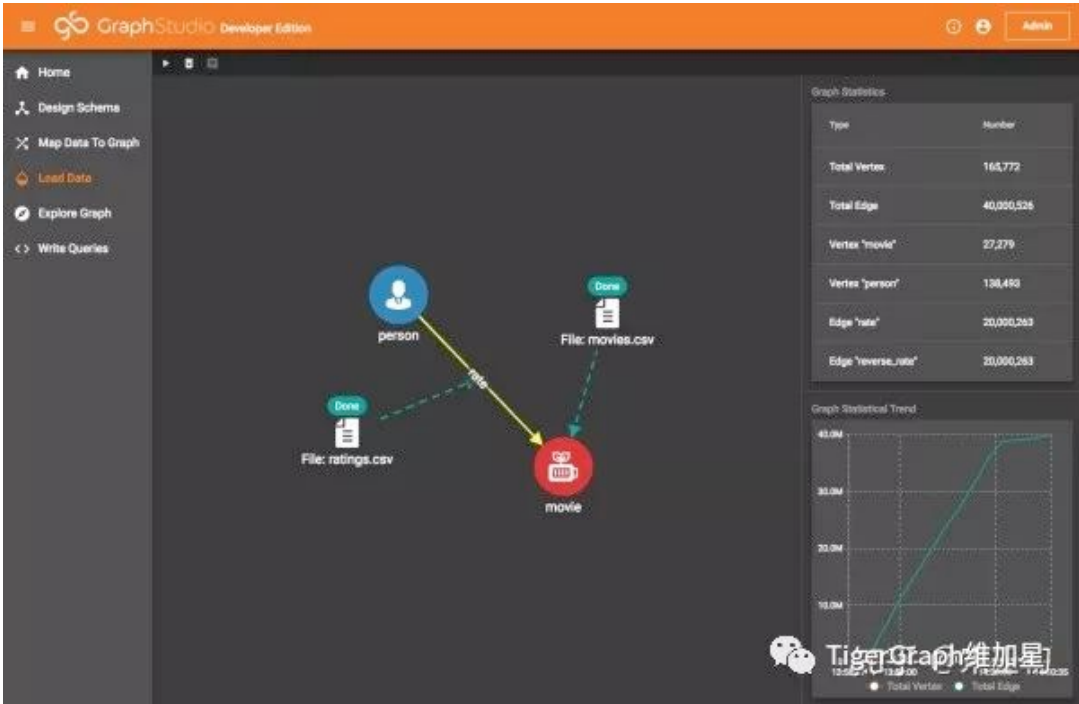
载入数据

接下来我们让TigerGraph系统根据我们创建的数据映射加载数据。点击左侧导航栏的Load Data项进入加载数据页面，点击工具栏中的开始加载按钮：



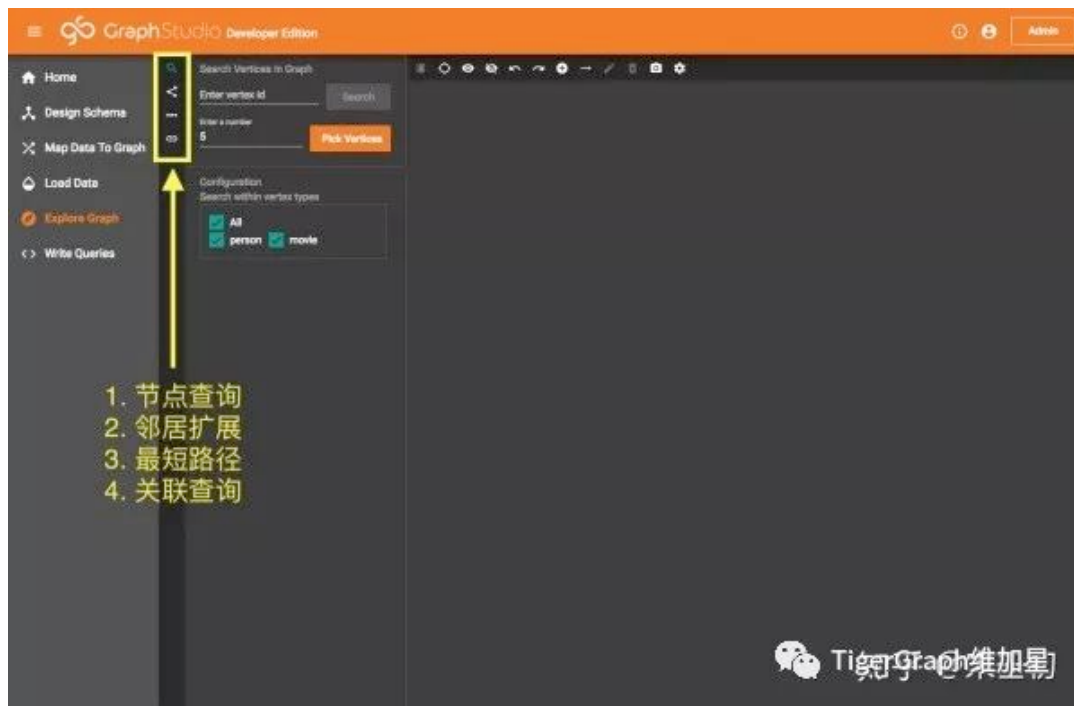
加载数据（Load Data）页面

加载整个数据集耗时仅2分钟，这仅仅是在个人苹果笔记本的虚拟机上就能达到的加载速度！加载完毕后我们通过右侧的统计信息看到总共加载了16.5万个节点和4000万条边。之前我们说总共有2000万条打分记录，而每条记录在TigerGraph图数据库中被加载到一条rate边和一条reverse_rate反向边，因此总共有4000万条边。



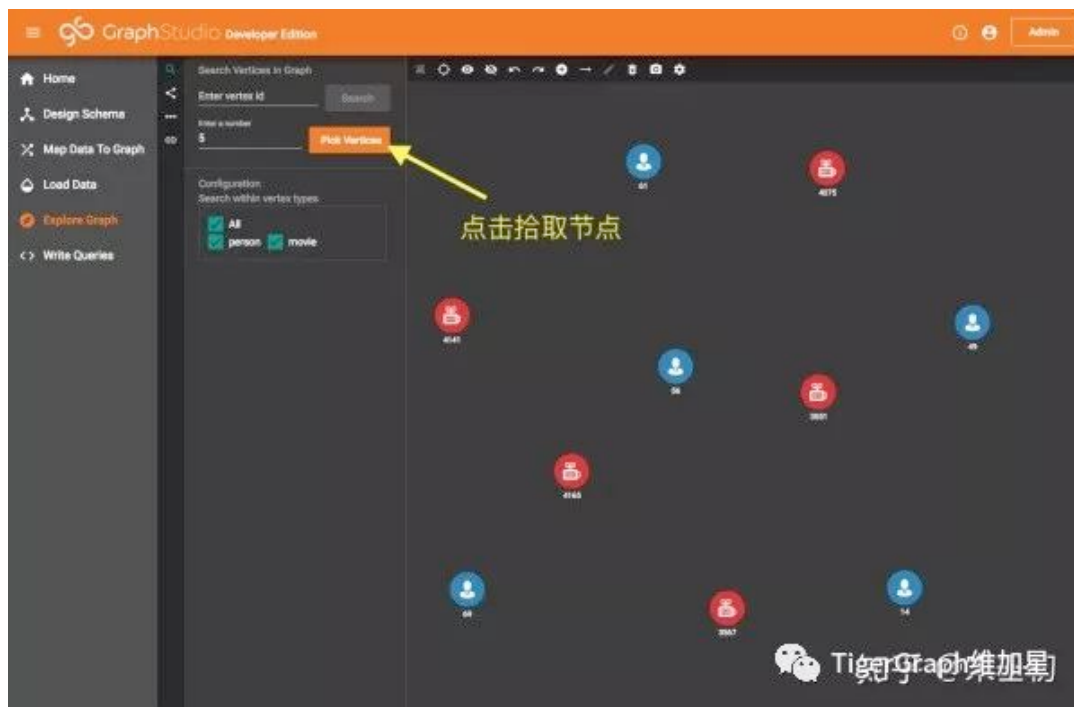
浏览图数据

下面我们利用GraphStudio内置的一些图数据浏览功能直观的感受一下刚刚加载的数据。点击左侧导航栏的Explore Graph项进入浏览图数据页面：



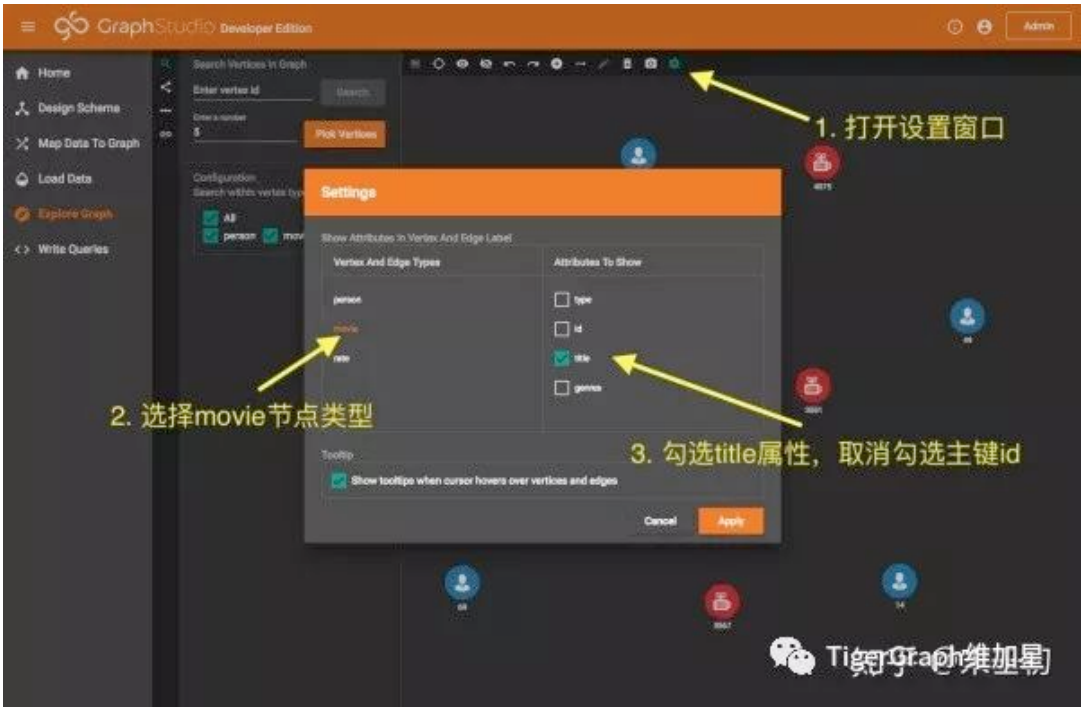
浏览图数据（Explore Graph）页面

首先我们点击拾取节点（Pick Vertices）按钮从图数据中拾取5个person节点和5个movie节点。这里的拾取不是随机的，因此每次拾取会返回相同的结果。如果你想要更多的节点，可以修改Enter a number中的数字。这里最大可以输入500。如果你知道节点的主键，可以在Enter vertex id输入框中输入主键的值，然后点击旁边的Search按钮拾取那个节点。配置（Configuration）可以控制拾取节点的类型范围，默认是从全部类型中拾取。你也可以勾选取消一些类型。



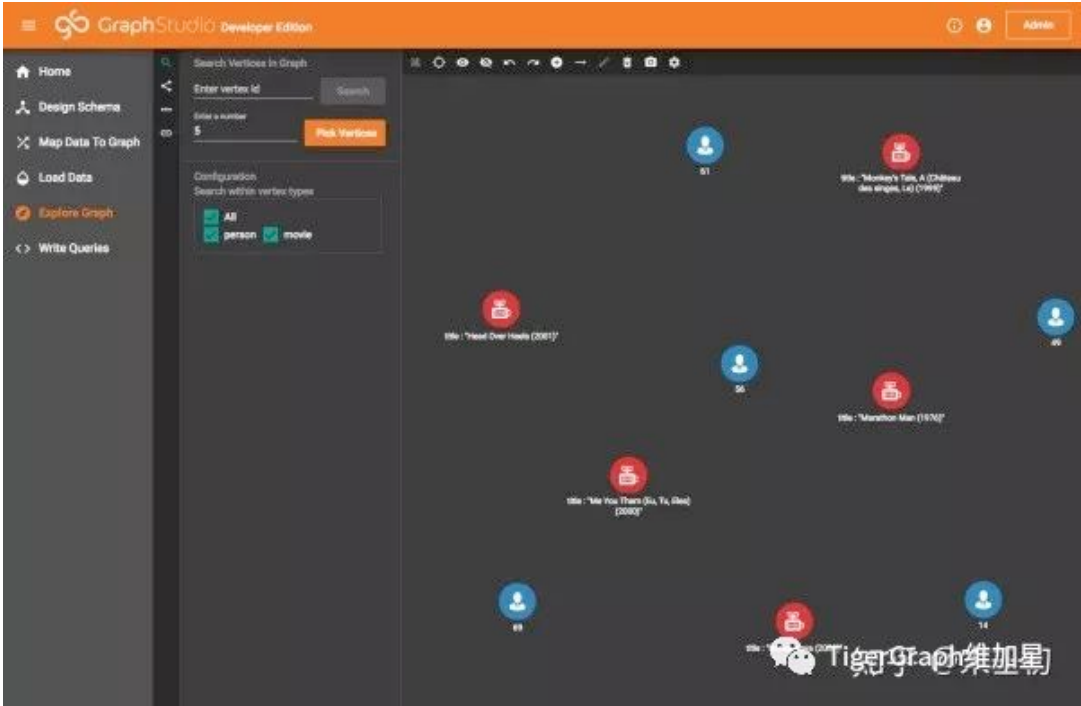
拾取5个person节点和movie节点

默认情况下所有节点显示的标签都是它们的主键。你可以修改设置显示其他属性，我们设置movie类型的节点显示它们的title属性：



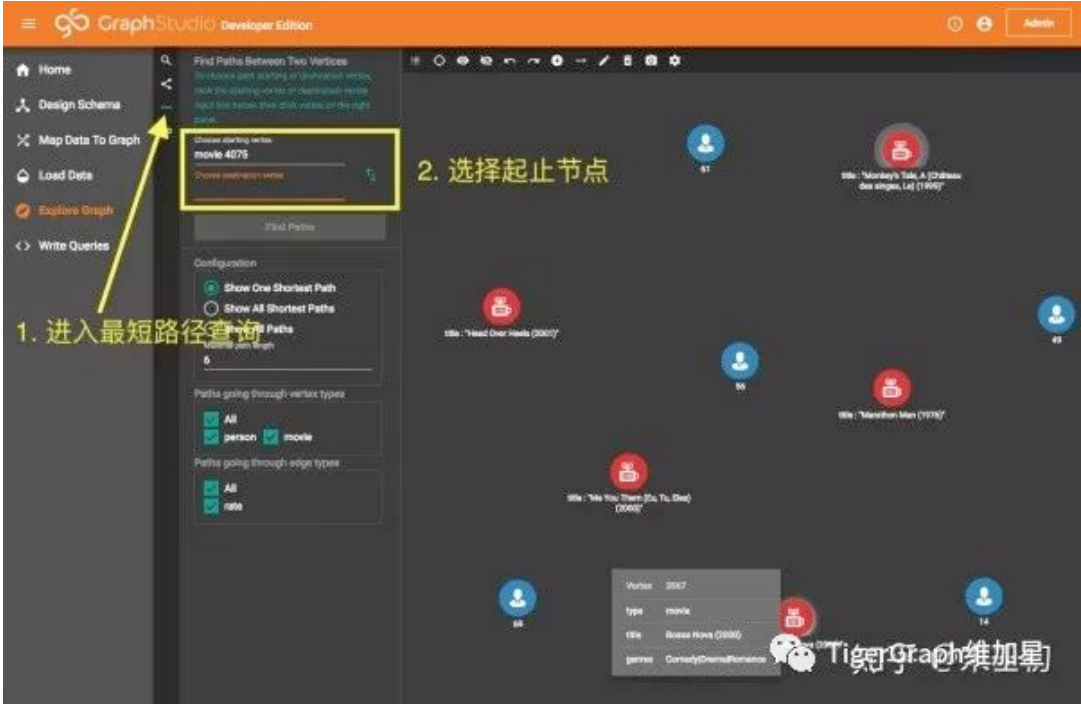
设置在movie类型节点的标签中显示title

完成修改之后，可以看到工作面板中的movie节点的标题被显示出来了，可视化变得更加直观。



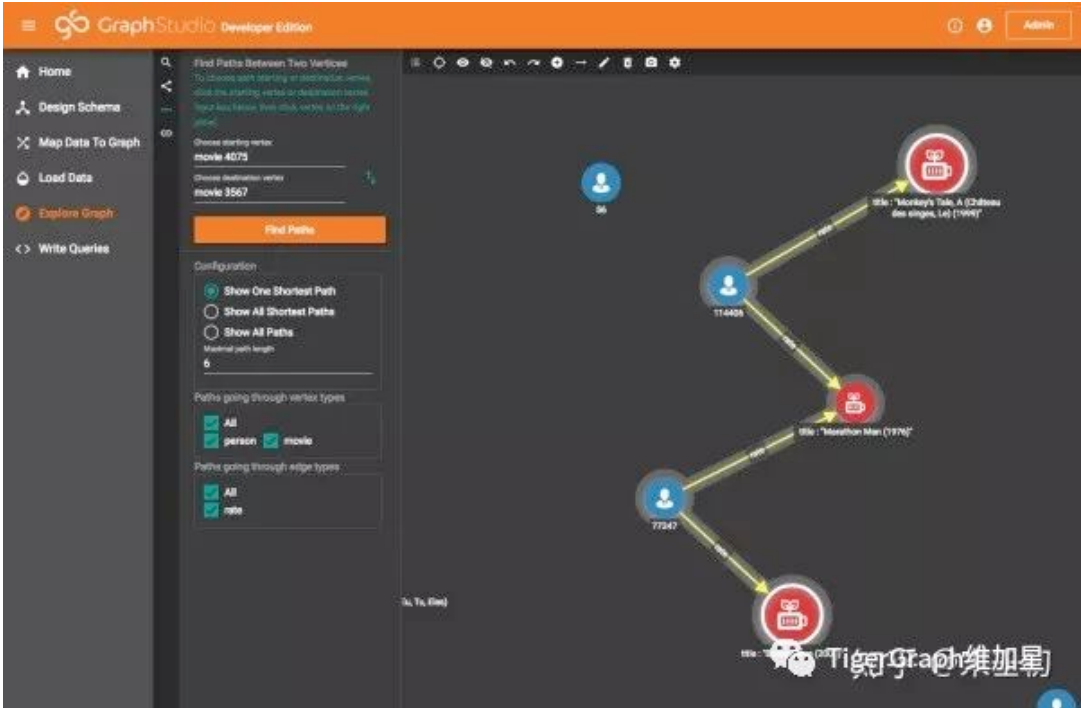
在movie节点上显示title

切换到黑色的纵向导航栏第三个最短路径项。点击选择起始节点（Choose starting vertex）输入框，再随意点击工作面板中的一个节点。再点击选择目标节点（Choose destination vertex）输入框，再随意选择工作面板中的另一个节点。



选择最短路径的起始节点和目标节点

点击查找路径（Find Paths）按钮，TigerGraph瞬间找到了两点之间的一条最短路径，长度为4。请尝试修改设置将打分显示在rate边的标签上~



两个电影节点之间的一条最短路径

你可以继续尝试浏览图数据页面的其他功能。相信你会通过一些操作发现这是一个非常密集的图（点与点之间有极多的连接路径）。

好了，现在你已经建立了图模型并成功载入了ml-20m数据集，并且通过浏览图数据直观的感受了person节点和movie节点是如何通过rate边互相连接的。你可能会惊奇的发现：哇塞，我居然没有写一行代码！

电影推荐算法实现

下面就是最激动人心的部分了～

你将很快用GSQL查询语言实现一个电影推荐的算法。

我们将实现引用资料[2]中的电影推荐算法。首先简单的介绍一下这个算法。

算法的输入为需要被推荐电影的person节点p，以及两个整数参数k1，k2。

算法的输出为一个movie节点集合，表示被推荐的电影，最多会推荐k2部电影。每个电影对应一个浮点数的推荐系数，系数越大则推荐越强烈。

算法的过程为：首先找到p打过分的所有电影集合m。然后找到所有对集合m中的电影打过分的人，计算这些人与p的看电影品味的相似度（余弦相似度），从中选出最相似的k1个人。找到这k1个人打过的电影中p没有打过的电影集合m2，计算这k1个人对m2中的电影打分的平均值，选择其中平均值最高的k2部电影推荐给p，将平均分作为推荐系数。

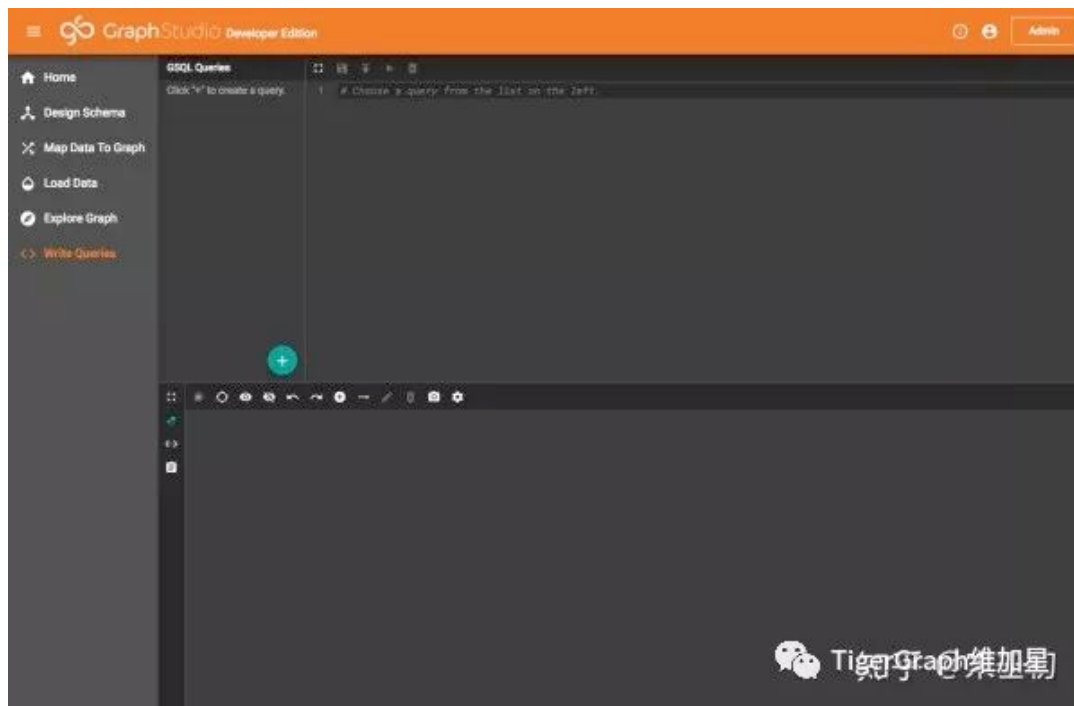
这里涉及到余弦相似度的概念。两个n维向量A和B的余弦相似度为：

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

来自维基百科

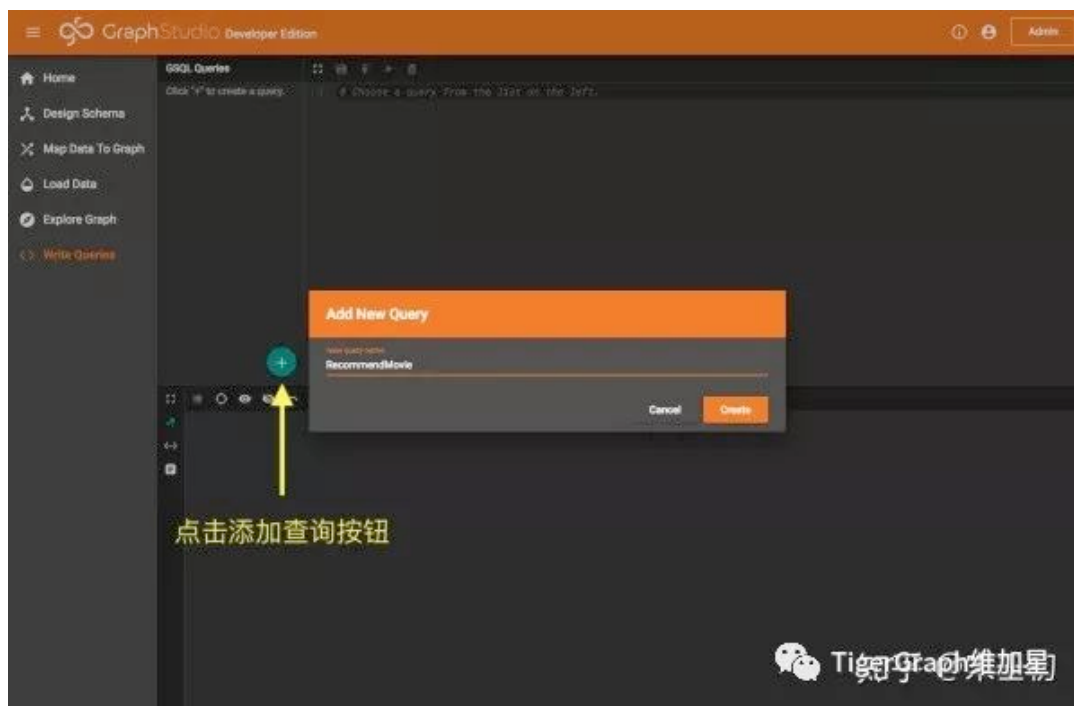
在电影推荐的例子中，给定两个人p1和p2，他们的品味相似度可以由两人共同打过的电影作为向量空间，由二人对这些电影的打分作为向量，通过计算两个向量的余弦相似度得到。由于所有的打分都在0.5到5之间（0.5分为一档），计算出来的相似度是一个(0, 1]的浮点数。

点击左侧导航栏的Write Queries项进入编写查询页面：



编写查询（Write Queries）页面

点击添加查询按钮，在弹出的窗口中输入查询名称：**RecommendMovie**。



添加查询

GraphStudio为我们生成了查询的模板。我们根据上文算法描述将查询的参数写为：

```
CREATE QUERY RecommendMovie(  
  vertex<person> p, int k1, int k2) FOR GRAPH MyGraph {  
  PSet = { p };}
```

它表示我们要通过与 p 品味最相似的 k_1 个人为 p 推荐 k_2 部电影。

在GSQL中，图运算的基本单元为从一个节点集合开始，经过由这些点、它们的出边

（**outgoing edges**）和出边的终端节点（**target vertex**）集合所构成的所有点-边-点的三元组集合上进行计算，并输出一个点集（可以是起始节点集或终端节点集）。在这个过程中可以在运行时在节点上绑定重载过 $=$ 和 $+=$ 运算符的累加器（**accumulator**）记录对三元组在起始节点集或终端节点集上进行**reduce**操作得到的结果。以上这段描述比较抽象，下面的例子中我们会进一步解释。

我们构建一个只包含 p 节点一个元素的节点集PSet:

```
CREATE QUERY RecommendMovie(
  vertex<person> p, int k1, int k2) FOR GRAPH MyGraph {
  PSet = { p };
```

然后，我们用下面的基本图计算单元找到 p 打过分的所有电影集合，并将其命名为**PRatedMovies**。我们可以看到这个基本图计算单元的输出是终端节点集 m 。这里的别名（ r 和 m ）是局部有效的，因此在其他基本图计算单元中可以被重用。

```
CREATE QUERY RecommendMovie(
  vertex<person> p, int k1, int k2) FOR GRAPH MyGraph {
  PSet = { p };

  PRatedMovies =
    SELECT m
    FROM PSet -(rate:r)-> movie:m;}
```

我们希望在**PRatedMovies**上打一个布尔的标记，表示 p 已经为这个电影打过分了，所以之后就不再推荐这个电影了。我们可以定义一个或累加器（**OrAccum**）命名为**@rated**，并在计算单元中的**ACCUM**子句中在 $movie$ 节点上打一个**true**的标记：

```
CREATE QUERY RecommendMovie(
  vertex<person> p, int k1, int k2) FOR GRAPH MyGraph {
  OrAccum @rated;

  PSet = { p };

  PRatedMovies =
    SELECT m
    FROM PSet -(rate:r)-> movie:m
    ACCUM m.@rated = true;}
```

接下来我们从**PRatedMovies**集合出发通过打分边的反向边（**reverse_rate**）找到所有对 p 打过分的人，并命名为**PeopleRatedSameMovies**：

```
CREATE QUERY RecommendMovie(
  vertex<person> p, int k1, int k2) FOR GRAPH MyGraph {
  ...

  PeopleRatedSameMovies =
    SELECT tgt
    FROM PRatedMovies:m -(reverse_rate:r)-> person:tgt
    WHERE tgt != p;}
```

下面我们计算**PeopleRatedSameMovies**集合中的每个**person**节点和**p**的余弦相似度。计算发生在第二个基本图计算单元里，需要知道**p**对每部电影的打分，所以我们修改第一个基本图计算单元，将**p**对电影的打分传递到**p**打过的电影上的一个叫**@ratingByP**的求和累加器里：

```
CREATE QUERY RecommendMovie(
  vertex<person> p, int k1, int k2) FOR GRAPH MyGraph {
  OrAccum @rated;
  SumAccum<double> @ratingByP;

  PSet = { p };

  PRatedMovies =
    SELECT m
    FROM PSet -(rate:r)-> movie:m
    ACCUM m.@rated = true, m.@ratingByP = r.rating;

  ...}
```

我们要计算**p**和**PeopleRatedSameMovies**集合中每一个人**q**的余弦相似度，回顾上文的余弦相似度公式，打分向量空间为**p**和**q**共同打过的电影，**p**对这些电影的打分向量为**A**，**q**的打分向量为**B**，那么我们可以定义4个新的求和累加器（**SumAccum**）：**@lengthA**，**@lengthB**，**@dotProductAB**和**@cosineSimilarity**，并且在第二个基本图计算单元的**ACCUM**子句中计算**@lengthA^2**，**@lengthB^2**和**@dotProductAB**：

```
CREATE QUERY RecommendMovie(
  vertex<person> p, int k1, int k2) FOR GRAPH MyGraph {
  ...
  SumAccum<double> @lengthA, @lengthB, @dotProductAB;
  SumAccum<double> @cosineSimilarity;

  ...

  PeopleRatedSameMovies =
    SELECT tgt
    FROM PRatedMovies:m -(reverse_rate:r)-> person:tgt
    WHERE tgt != p
    ACCUM tgt.@lengthA += m.@ratingByP * m.@ratingByP,
          tgt.@lengthB += r.rating * r.rating,
          tgt.@dotProductAB += m.@ratingByP * r.rating;}
```

然后，我们在POST-ACCUM子句中完成@cosineSimilarity的计算。由于POST-ACCUM是在节点集合三元组所有的累加操作reduce完成之后，这时候tgt上面已经拥有了最终的@lengthA^2, @lengthB^2和@dotProductAB的值了：

```
CREATE QUERY RecommendMovie(
  vertex<person> p, int k1, int k2) FOR GRAPH MyGraph {
  ...

  PeopleRatedSameMovies =
    ...
    POST-ACCUM
      tgt.@cosineSimilarity =
        tgt.@dotProductAB / sqrt(tgt.@lengthA) / sqrt(tgt.@lengthB);}
```

我们通过ORDER BY子句和LIMIT子句选择与p品味最相似的k1个人：

```
CREATE QUERY RecommendMovie(
  vertex<person> p, int k1, int k2) FOR GRAPH MyGraph {
  ...

  PeopleRatedSameMovies =
    ...
    ORDER BY tgt.@cosineSimilarity DESC
    LIMIT k1;}
```

接下来，我们从PeopleRatedSameMovies集合出发，找到这些人打过分并且p没有打过分的电影集合：

```
CREATE QUERY RecommendMovie(
  vertex<person> p, int k1, int k2) FOR GRAPH MyGraph {
  ...

  RecommendedMovies =
    SELECT m
    FROM PeopleRatedSameMovies -(rate:r)-> movie:m
    WHERE m.@rated == false;}
```

我们添加一个平均值累加器（AvgAccum）名为@recommendScore用来累计这k1个人对这些电影的平均打分，并且根据打分从高到低排序，并选出其中得分最高的k2部电影作为推荐结果：

```
CREATE QUERY RecommendMovie(
  vertex<person> p, int k1, int k2) FOR GRAPH MyGraph {
  ...
  AvgAccum @recommendScore;

  ...}
```



```
RecommendedMovies =
  SELECT m
  FROM PeopleRatedSameMovies -(rate:r)-> movie:m
  WHERE m.@rated == false
  ACCUM m.@recommendScore += r.rating
  ORDER BY m.@recommendScore DESC
  LIMIT k2;}
```

最后，我们输出RecommendedMovies的title和@recommendScore作为结果：

```
CREATE QUERY RecommendMovie(
  vertex<person> p, int k1, int k2) FOR GRAPH MyGraph {
  ...

  PRINT RecommendedMovies.title,
    RecommendedMovies.@recommendScore;}
```

耶！我们完成了整个查询！来看一下完整的查询吧：

```
CREATE QUERY RecommendMovie(
  vertex<person> p, int k1, int k2) FOR GRAPH MyGraph {
  OrAccum @rated;
  SumAccum<double> @ratingByP;
  SumAccum<double> @lengthA, @lengthB, @dotProductAB;
  SumAccum<double> @cosineSimilarity;
  AvgAccum @recommendScore;

  PSet = { p };

  PRatedMovies =
    SELECT m
    FROM PSet -(rate:r)-> movie:m
    ACCUM m.@rated = true, m.@ratingByP = r.rating;

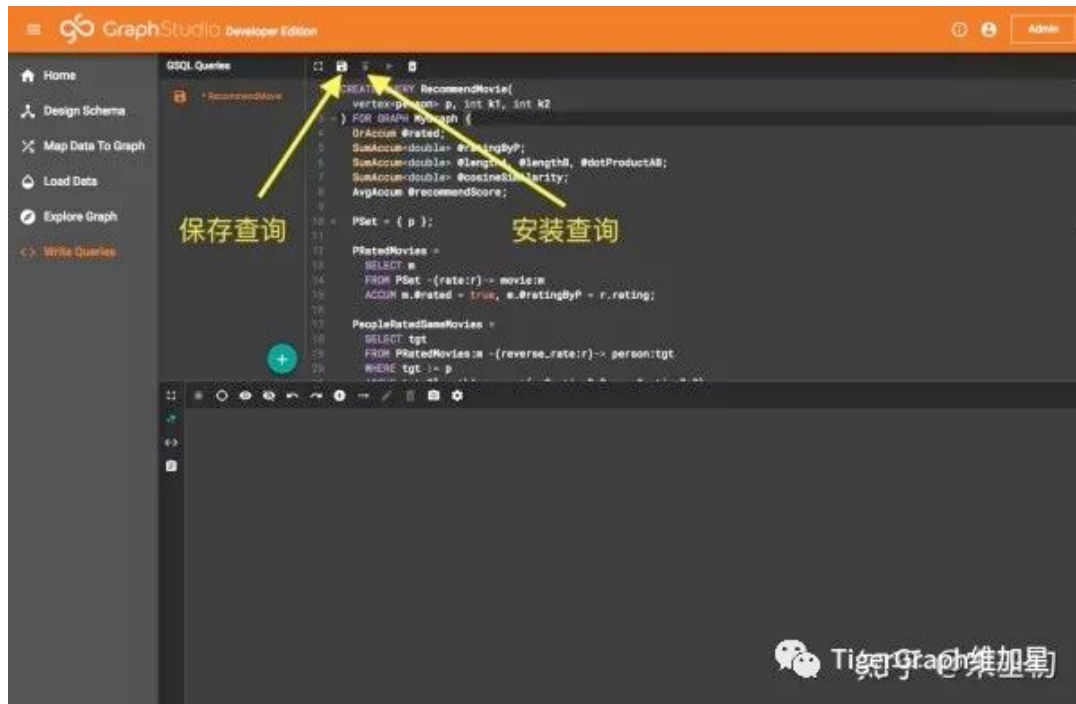
  PeopleRatedSameMovies =
    SELECT tgt
    FROM PRatedMovies:m -(reverse_rate:r)-> person:tgt
    WHERE tgt != p
    ACCUM tgt.@lengthA += m.@ratingByP * m.@ratingByP,
          tgt.@lengthB += r.rating * r.rating,
          tgt.@dotProductAB += m.@ratingByP * r.rating
    POST-ACCUM
      tgt.@cosineSimilarity =
        tgt.@dotProductAB / sqrt(tgt.@lengthA) / sqrt(tgt.@lengthB)
    ORDER BY tgt.@cosineSimilarity DESC
    LIMIT k1;

  RecommendedMovies =
    SELECT m
    FROM PeopleRatedSameMovies -(rate:r)-> movie:m
    WHERE m.@rated == false
    ACCUM m.@recommendScore += r.rating
```

```
ORDER BY m.@recommendScore DESC
LIMIT k2;

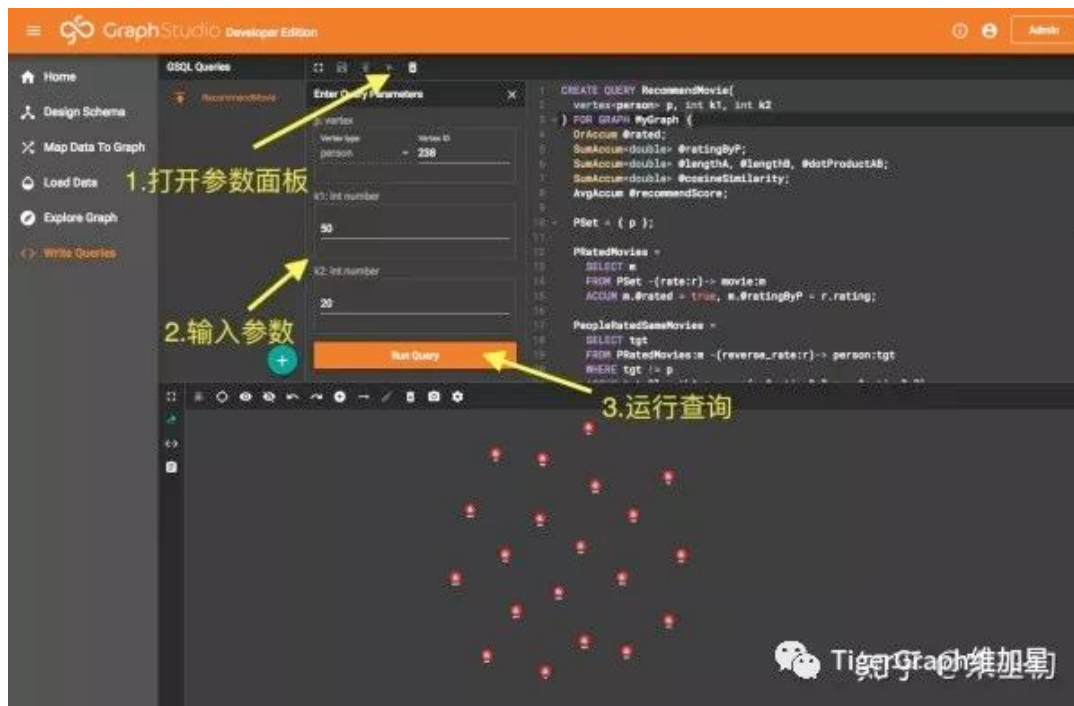
PRINT RecommendedMovies.title,
      RecommendedMovies.@recommendScore;}
```

好了，我们点击工具栏上的保存按钮保存查询，然后点击安装查询按钮安装查询。整个安装大约需要一分钟：



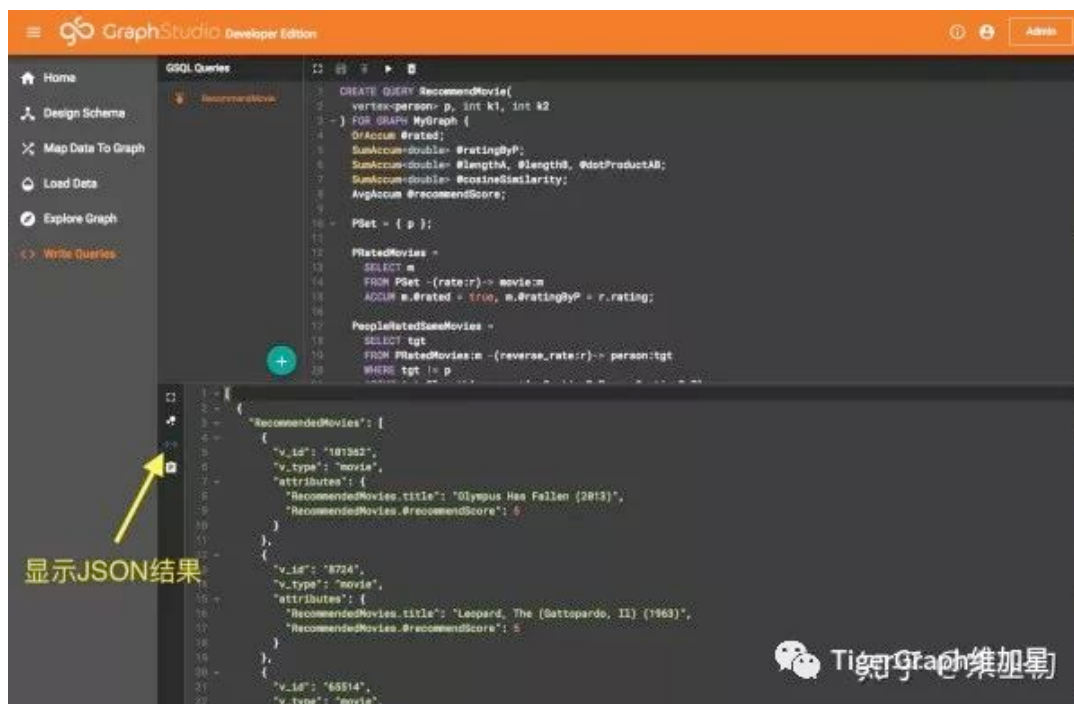
保存并安装查询

安装成功后，点击工具栏上的运行查询按钮，由于这个查询需要输入参数，参数面板会弹出。输入一个人的id（比如238），给定k1(比如50)和k2（比如20），然后点击参数面板下方的运行查询按钮，很快结果面板中显示了推荐的电影：



运行查询

点击结果面板左侧的显示JSON项，可以查看JSON格式的查询返回结果：



显示JSON结果

厉害了，你已经搭建完成了一个简单的电影推荐系统！接下来，你可以通过TigerGraph系统提供的RESTFul接口把这个推荐系统和你的前端的应用直接进行集成。

TigerGraph的默认RESTFul服务端口为9000，安装的GSQL查询全部可以通过http GET的方式访问，只要在URL中加入查询的参数。下面这个curl命令就是一个调用的例子（将MACHINE-IP替换为你的安装TigerGraph系统的机器的IP）

```
curl 'http://<MACHINE-IP>:9000/query/MyGraph/RecommendMovie?p=238&k1=50&k2=20'
```

总结

本文介绍了如何使用TigerGraph图数据库快速搭建一个简单的电影推荐系统。由于篇幅限制，这里只涉及到了部分TigerGraph图数据库的功能，很多核心功能如实时数据更新以及GSQL强大而全面的语法都没有涉及，GraphStudio中很多更加酷炫的功能也没有演示。至于而本文采用的推荐算法也是非常朴素的。如果这引起了你的兴趣，欢迎阅读TigerGraph开发文档，以及拓展阅读中的其他资料！

企业版的TigerGraph图数据库支持分布式、在线图模式变更（online schema change）、多图（mutiple graphs）以及众多企业级功能。关于企业版与开发者版本的区别，详见[这里](#)：Download and Compare TigerGraph Editions, Experience the Real-Time Graph Database First Hand

引用资料：

- F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=<The MovieLens Datasets>
- 字凤芹, 牛进, 毕柱兰, 沈加敏. 基于图数据库的电影推荐系统设计. 软件导刊. 2016(1):144-6.

拓展阅读：


- [性能炸裂！图数据库TigerGraph开发者版本入门](#)
- [跨跨越鸿沟——图数据库查询语言的八个先决条件——上篇](#)
- [跨越鸿沟——GSQL如何解决图数据库查询语言八大先决条件？](#)
- [TigerGraph GSQL 入门](#)

[阅读原文](#)



微信扫一扫

关注该公众号

Proudly powered by  WordPress

