ALIBABA CLOUD

阿里云

图数据库
使用范例

文档版本：20220406

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.银行、金融

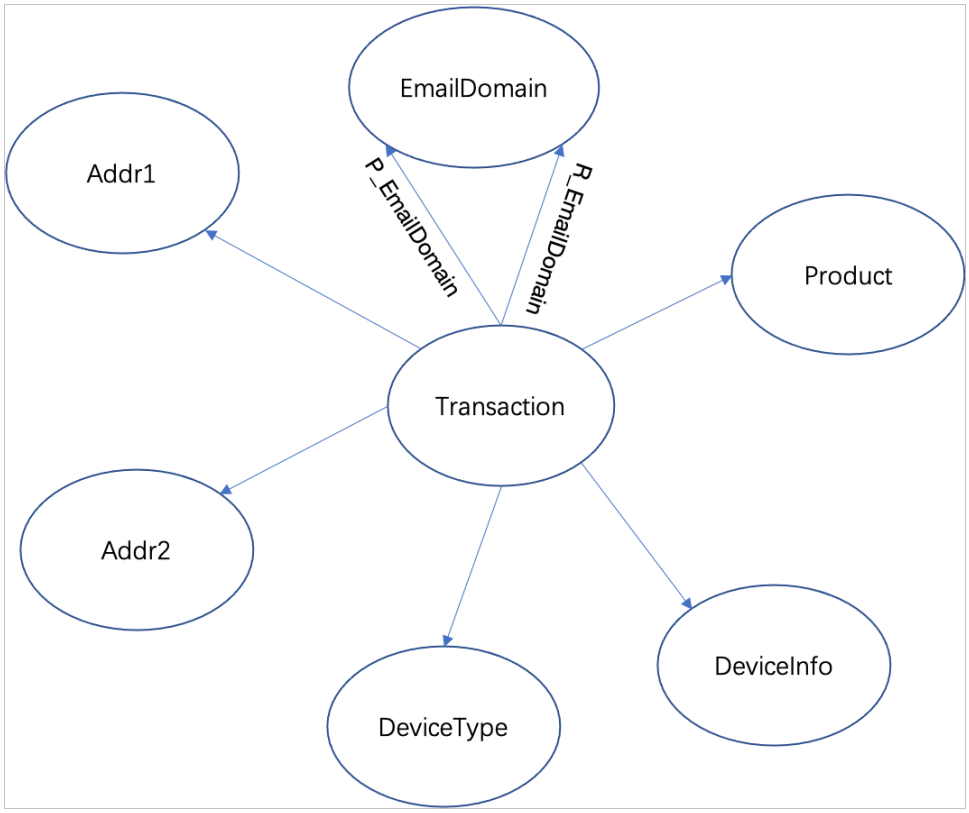
在现代欺诈和各类金融犯罪中，欺诈者通过改变自身身份等达到逃避风控规则的欺诈目的。您可以通过图数据库GDB建立跟踪用户行为的图结构，实时分析欺诈行为的离散数据，识别欺诈环，帮助您快速防范和解决欺诈行为。

1、数据模型

以金融交易领域公开数据集IEEE-CIS Fraud Detection数据集为例。更多信息，请参见数据模型参考下载。

数据是由电商平台Vesta提供的交易记录，包括了交易相关的设备、地址、邮箱等信息，可以将数据模型抽象为下图：

> **说明** 在本文示例中，需要对交易涉及到属性信息进行较多的过滤、统计等操作，所以这里将交易记录及其属性信息建模成边；如果在实际的业务中更关注交易记录本身，可以考虑将交易记录的属性信息直接建模成点的属性。



示例数据如下：

- 点文件：

```
//Transaction交易信息。 ~id,~label,is_fraud:int 2987000,"transaction",0 2987001,"transaction",0 2987002,"transaction",0 2987003,"transaction",0 2987004,"transaction",0 //DeviceType设备类型信息。 ~id,~label "mobile","devicetype" "desktop","devicetype"
```

- 边文件：

```
~id,~from,~to,~label 1,2987000,"","t_e_p" 2,2987000,"","t_e_r" 3,2987000,"315.0","t_a1" 4,2987000,"87.0","t_a2"
```

2、创建实例

创建图数据库实例。具体操作，请参见创建主实例。

> 说明
>
> - 实例购买成功后，您可以在**实例列表**页面查看实例相关信息。通常，实例创建成功需要3~5分钟。
> - 创建GDB实例后，您需要创建账号和密码、设置安全组，保证您具有GDB实例的访问权限。具体操作，请参见创建账号和设置白名单。

3、数据导入

图数据库GDB支持从多种数据源将数据导入至图数据库GDB，您可以使用以下两种方式进行数据导入：

- 从阿里云对象存储OSS导入数据至图数据库GDB实例。
- 使用DataWorks导入数据至图数据库GDB实例。

4、连接实例

图数据库GDB支持多种方法连接实例，您可以通过以下五种方式连接实例：

- 通过GDB控制台直接登录数据管理服务DMS，更加方便快捷地远程访问、在线管理您的GDB数据库。具体操作，请参见通 过DMS登录GDB数据库。
- 通过开源组件GDB Console可视化控制台登录图数据库，可视化界面，操作简单，并可根据业务需求对可视化界面进行二次开发。具体操作，请参见通过开源组件GDB Console登录图数据库。
- 通过Gremlin Console连接实例，命令行模式，适合Gremlin内核版本，适合用于查询语句性能优化。具体操作，请参见通过Gremlin Console连接实例。
- 通过Cypher Shell连接实例，命令行模式，适合Cypher内核版本。具体操作，请参见通过Cypher Shell连接实例 。
- 通过SDK连接，支持Java、Python、.Net、Go、Node.js五种SDK。具体操作，请参见SDK参考。

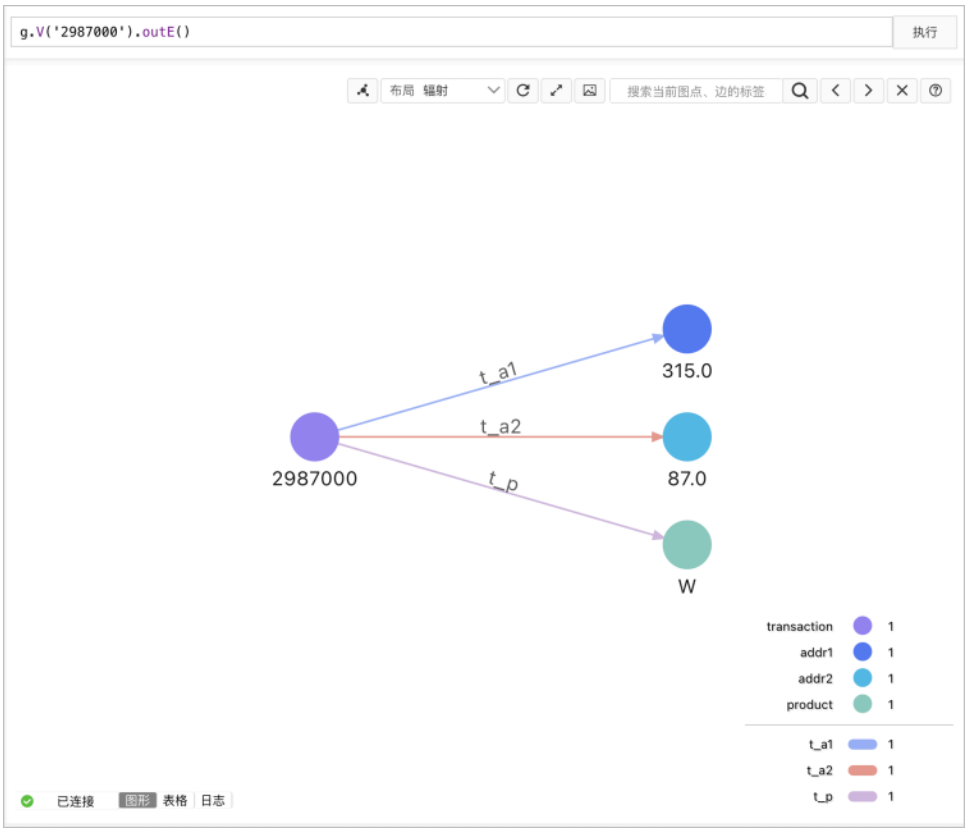
5、使用范例

- 简单查询
  - 数据统计：

```
//统计点的数目。 gremlin> g.V().count() ==>592789 //统计边的数目。 gremlin> g.E().count()
==>2533038
```

- 过滤查询、排序查询：

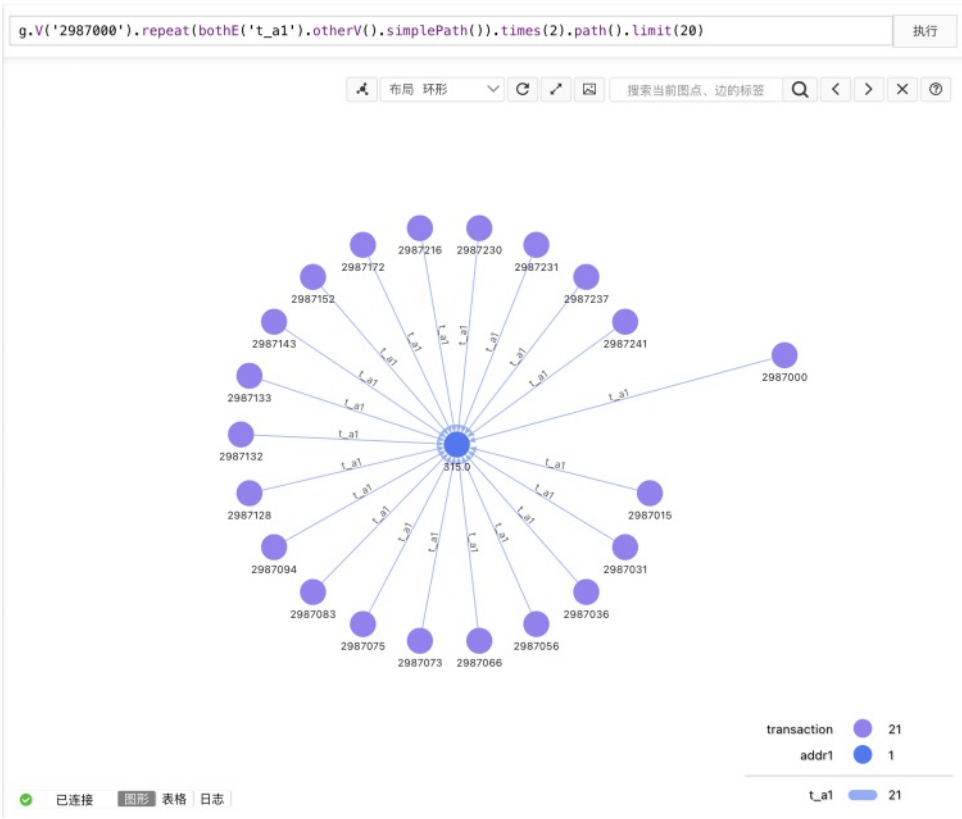
```
//查询ID为2987000的交易信息。 gremlin> g.V('2987000').valueMap(true) ==>[id:2987000,label
:transaction,is_fraud:[0]] //查询ID为2987000的交易信息。 gremlin> g.V('2987000').outE() =
=>e[3][2987000-t_a1->315.0] ==>e[4][2987000-t_a2->87.0] ==>e[5][2987000-t_p->W]
```



- 通用场景

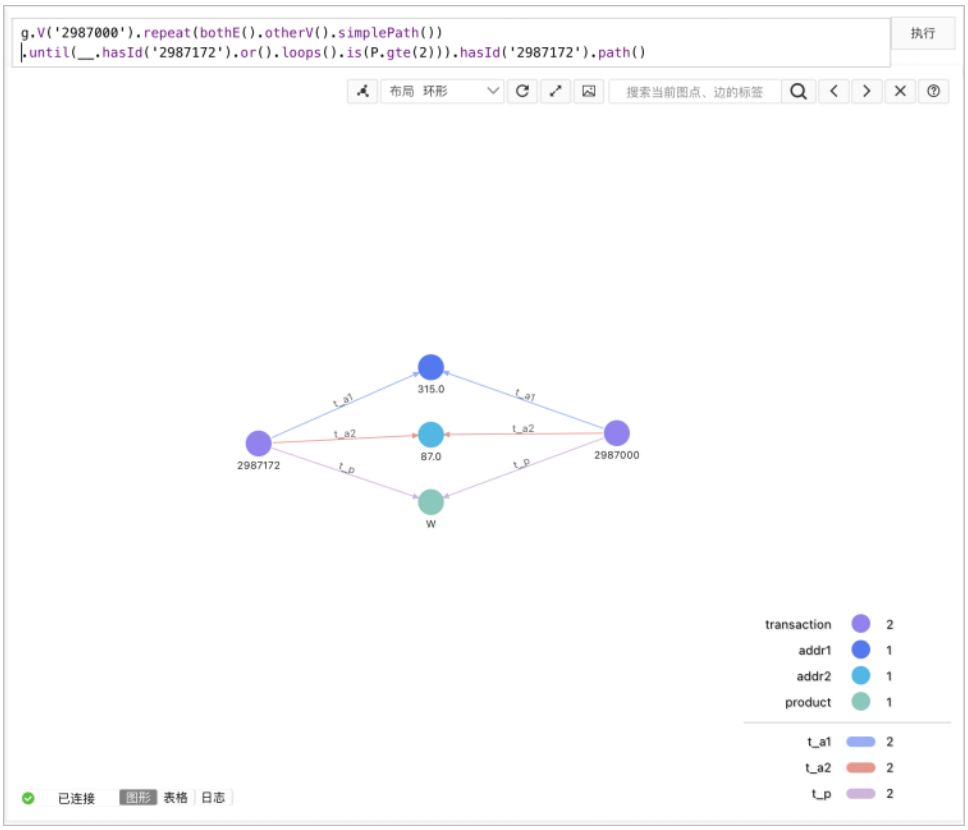
- K阶邻居：

```
//查询和交易记录2987000有相同地址的其他交易记录。 gremlin> g.V('2987000').repeat(bothE('t_a1').otherV().simplePath()).times(2).path() //bothE() 部分控制查询边类型，times()部分控制查询深度。 ==>[v[2987000],e[3][2987000-t_a1->315.0],v[315.0],e[86][2987015-t_a1->315.0],v[2987015]] ==>[v[2987000],e[3][2987000-t_a1->315.0],v[315.0],e[171][2987031-t_a1->315.0],v[2987031]] ==>[v[2987000],e[3][2987000-t_a1->315.0],v[315.0],e[196][2987036-t_a1->315.0],v[2987036]] ==>[v[2987000],e[3][2987000-t_a1->315.0],v[315.0],e[305][2987056-t_a1->315.0],v[2987056]] ==>[v[2987000],e[3][2987000-t_a1->315.0],v[315.0],e[356][2987066-t_a1->315.0],v[2987066]] ==>[v[2987000],e[3][2987000-t_a1->315.0],v[315.0],e[399][2987073-t_a1->315.0],v[2987073]] ==>[v[2987000],e[3][2987000-t_a1->315.0],v[315.0],e[411][2987075-t_a1->315.0],v[2987075]] ==>[v[2987000],e[3][2987000-t_a1->315.0],v[315.0],e[451][2987083-t_a1->315.0],v[2987083]] ==>[v[2987000],e[3][2987000-t_a1->315.0],v[315.0],e[510][2987094-t_a1->315.0],v[2987094]] ......
```



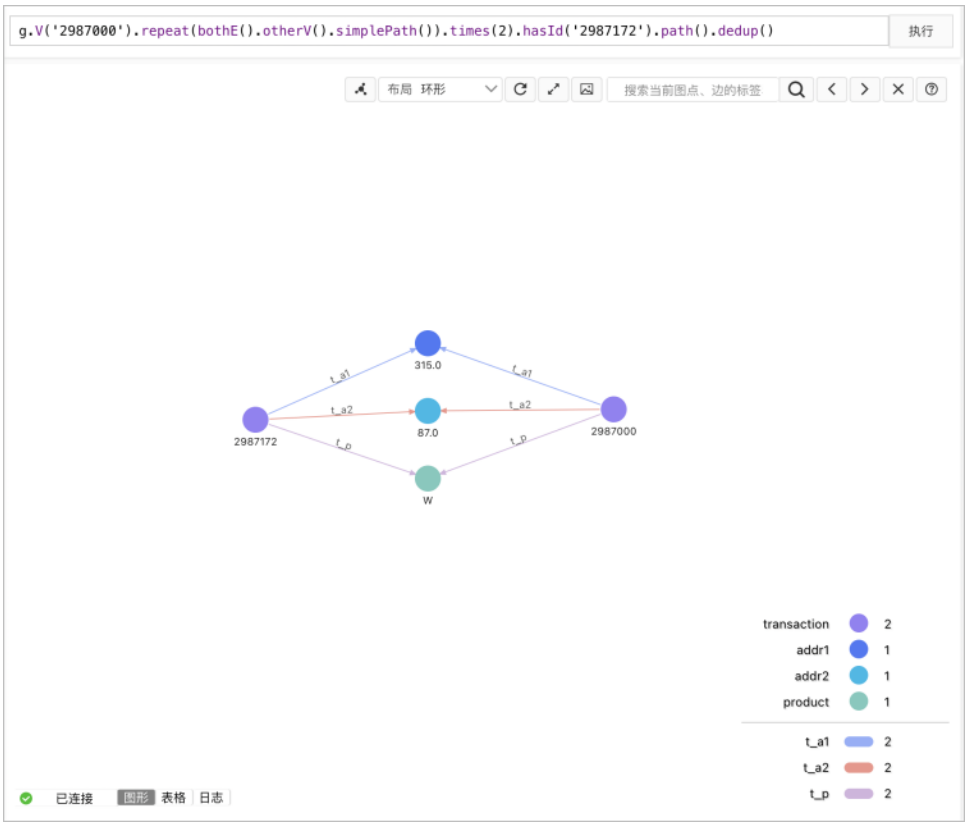
- 最短路径：

```
//查询交易记录2987000和交易记录2987172的最短路径，最大深度为2。 gremlin> g.V('2987000').repeat(bothE().otherV().simplePath()) .until(hasId('2987172').or().loops().is(gt(2L))) //hasId()部分控制结束ID，gt()部分查询深度。 .hasId('2987172').path().dedup() ==>[v[2987000],e[3][2987000-t_a1->315.0],v[315.0],e[938][2987172-t_a1->315.0],v[2987172]] ==>[v[2987000],e[4][2987000-t_a2->87.0],v[87.0],e[939][2987172-t_a2->87.0],v[2987172]] ==>[v[2987000],e[5][2987000-t_p->W],v[W],e[940][2987172-t_p->W],v[2987172]]
```



- 共同邻居：

```
//查询交易记录2987000和交易记录2987172的共同邻居，从而找到具备相同属性（地址、设备）的交易记录。
gremlin> g.V('2987000').repeat(bothE().otherV().simplePath()).times(2).hasId('2987172')
.path().dedup() //hasId()部分控制结束ID。 ==>[v[2987000],e[3][2987000-t_a1->315.0],v[315.
0],e[938][2987172-t_a1->315.0],v[2987172]] ==>[v[2987000],e[4][2987000-t_a2->87.0],v[87
.0],e[939][2987172-t_a2->87.0],v[2987172]] ==>[v[2987000],e[5][2987000-t_p->W],v[W],e[9
40][2987172-t_p->W],v[2987172]]
```



- Jaccard相似度：

```
//查询和交易记录2987000相似的交易记录，根据相似度分数取前10。 gremlin> g.V('2987000') .sideEf
fect(out().store('v1n')) .as('v1') .select('v1n').unfold().in().limit(100).simplePath()
.dedup().as('v2') .project('i', 'u') .by(select('v2').out().where(within('v1n')).count(
)) .by(union(select('v2').out().fold(),select('v1n')).unfold().dedup().count()) .projec
t('Transaction', '相似度') .by(select('v2').id()) .by(math('i/u')) .order().by(select('
相似度'), desc).limit(10) ==>[Transaction:2987015,相似度:1.0] ==>[Transaction:2987056,相
似度:1.0] ==>[Transaction:2987216,相似度:1.0] ==>[Transaction:2987251,相似度:1.0] ==>[Tra
nsaction:2987371,相似度:1.0] ==>[Transaction:2987474,相似度:1.0] ==>[Transaction:2987699
,相似度:1.0] ==>[Transaction:2987707,相似度:1.0] ==>[Transaction:2987735,相似度:1.0] ==>[
Transaction:2987770,相似度:1.0]
```

6、客户效果

某头部商业银行信用卡中心，自动构建890个特征，发现嫌欺诈商户12个、预测逾期用户1.86万人、套现卡片2.2万张、识别欺诈团伙102个，模型召回命中率92.4%（客户模型为60%）。

2.社交网络

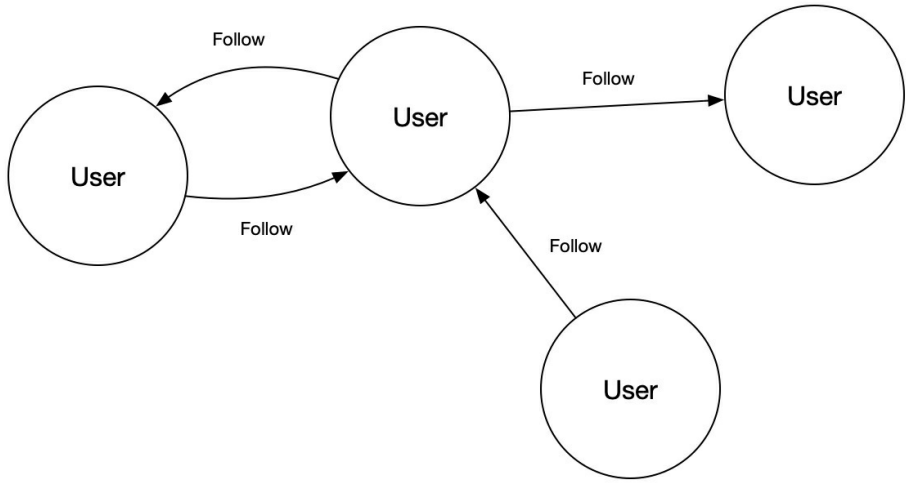
社交类业务场景的数据模型天然具备高度连接的特点，图数据库GDB可以为社交类业务提供天然的图模型支持，更加完美的匹配和理解您的数据。使用图数据库GDB，可以显著提升社交类业务程序的开发效率和质量，减少数据模型转换带来的额外损耗。

1、数据模型

以社交领域公开数据集Twitter社交关系为例。更多信息，请参见数据模型参考下载。

图数据库GDB使用属性图模型来表示和处理数据，可以将数据模型抽象为下图：

> 说明
> - User：用户节点。
> - Follow：代表单向的好友关系。比如关注，或者是对方的粉丝。



示例数据如下：

- 点文件：

```
~id,name:string 1,a0e05a4a-65c0-11e9-a5ce-00163e0416f8 2,a0e05e28-65c0-11e9-a5ce-00163e0416f8 3,a0e05f86-65c0-11e9-a5ce-00163e0416f8 4,a0e0606c-65c0-11e9-a5ce-00163e0416f8 5,a0e0613e-65c0-11e9-a5ce-00163e0416f8 6,a0e06206-65c0-11e9-a5ce-00163e0416f8
```

- 边文件：

```
~id,~from,~to,weight:double 212416660,1116299,4377946,0.443259 212416661,1116300,4377946,0.0303036 212416662,181406,4377946,0.753659 212416663,4084735,4377946,0.991974 212416664,1937755,4377946,0.79248
```

2、创建实例

创建图数据库实例。具体操作，请参见创建主实例。

> 说明
>
> - 实例购买成功后，您可以在**实例列表**页面查看实例相关信息。通常，实例创建成功需要3~5分钟。
> - 创建GDB实例后，您需要创建账号和密码、设置安全组，保证您具有GDB实例的访问权限。具体操作，请参见创建账号和设置白名单。

3、数据导入

图数据库GDB支持从多种数据源将数据导入至图数据库GDB，您可以使用以下两种方式进行数据导入：

- 从阿里云对象存储OSS导入数据至图数据库GDB实例。
- 使用DataWorks导入数据至图数据库GDB实例。

4、连接实例

图数据库GDB支持多种方法连接实例，您可以通过以下五种方式连接实例：

- 通过GDB控制台直接登录数据管理服务DMS，更加方便快捷地远程访问、在线管理您的GDB数据库。具体操作，请参见通 过DMS登录GDB数据库。
- 通过开源组件GDB Console可视化控制台登录图数据库，可视化界面，操作简单，并可根据业务需求对可视化界面进行二次开发。具体操作，请参见通过开源组件GDB Console登录图数据库。
- 通过Gremlin Console连接实例，命令行模式，适合Gremlin内核版本，适合用于查询语句性能优化。具体操作，请参见通过Gremlin Console连接实例。
- 通过Cypher Shell连接实例，命令行模式，适合Cypher内核版本。具体操作，请参见通过Cypher Shell连接实例 。
- 通过SDK连接，支持Java、Python、.Net、Go、Node.js五种SDK。具体操作，请参见SDK参考。

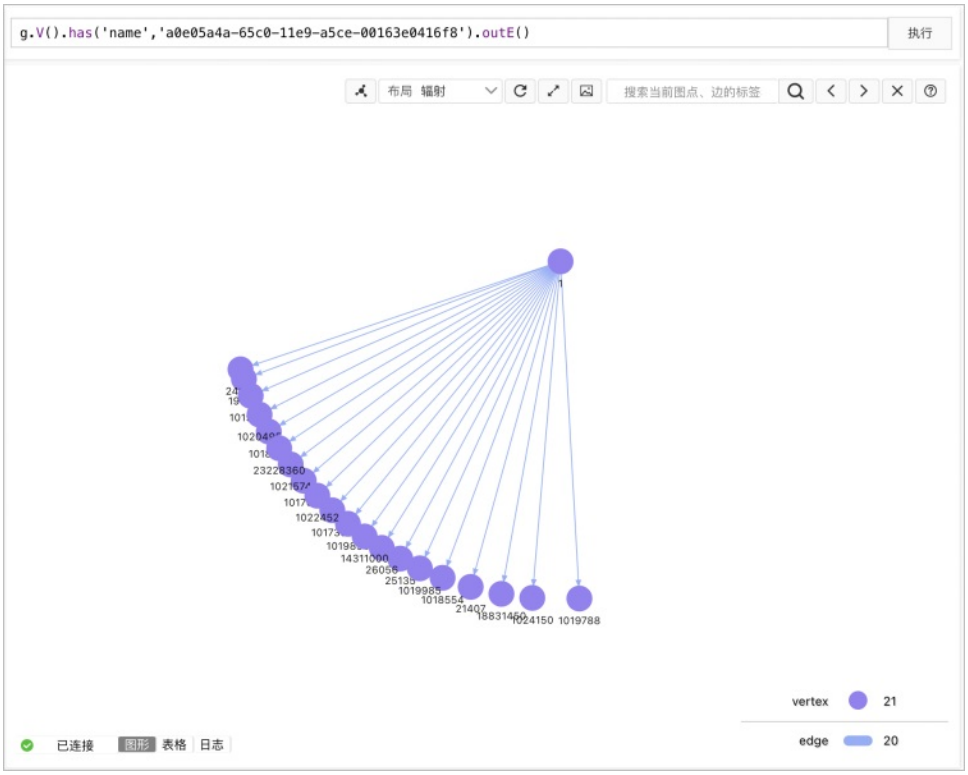
5、使用范例

- 简单查询
  - 数据统计：

```
//统计点的数目 gremlin> g.V().count() ==>41999999 //统计边的数目 gremlin> g.E().count() ==>22191165
```

  - 过滤查询、排序查询：

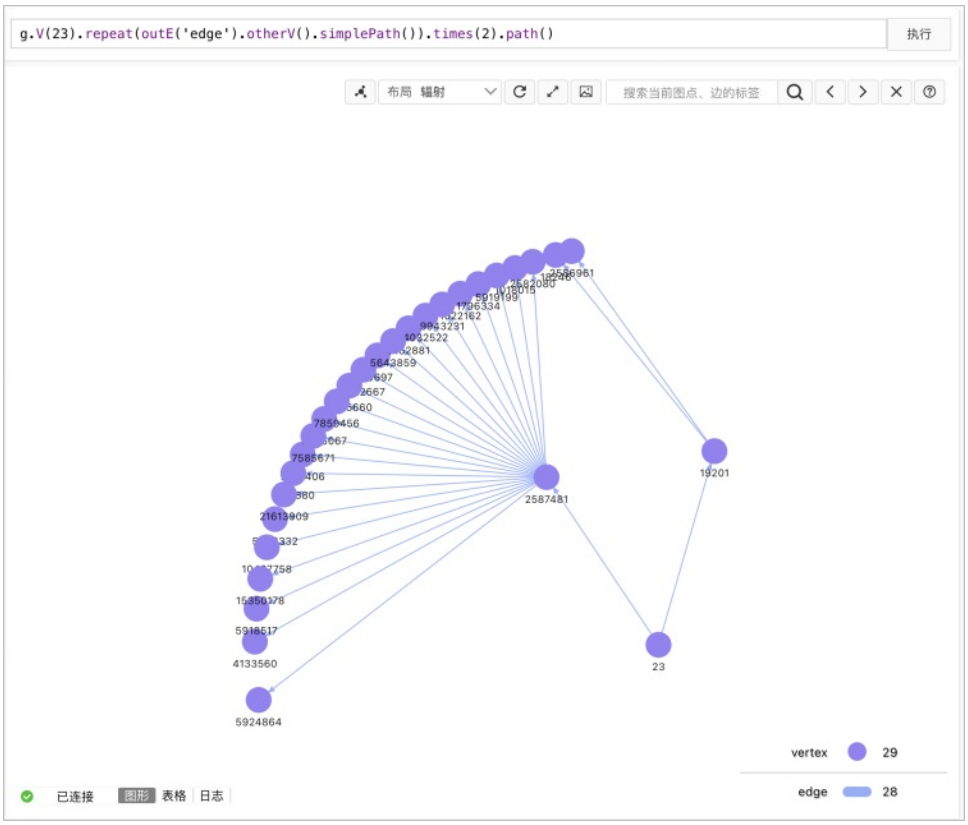
```
//查询name为a0e05a4a-65c0-11e9-a5ce-00163e0416f8的用户。 gremlin> g.V().has('name','a0e05a4a-65c0-11e9-a5ce-00163e0416f8').valueMap(true) //条件查询，类似select ... where ...，根据业务修改has()中的内容即可。 ==>[id:1,label:vertex,name:[a0e05a4a-65c0-11e9-a5ce-00163e0416f8]] //查询name为a0e05a4a-65c0-11e9-a5ce-00163e0416f8的用户的关注列表。 gremlin> g.V().has('name','a0e05a4a-65c0-11e9-a5ce-00163e0416f8').outE().valueMap(true) //条件查询，类似select ... where ...，根据业务修改has()中的内容即可 ==>[id:217089344,label:edge,weight:0.769055] ==>[id:220429042,label:edge,weight:0.290449] ==>[id:227652991,label:edge,weight:0.962171] ==>[id:234881614,label:edge,weight:0.0887247] ==>[id:250193757,label:edge,weight:0.756271] ==>[id:252223359,label:edge,weight:0.990445] ==>[id:252494754,label:edge,weight:0.494867] ==>[id:254012304,label:edge,weight:0.788503] ==>[id:260893506,label:edge,weight:0.247677] ==>[id:228404583,label:edge,weight:0.0742597] ==>[id:243912806,label:edge,weight:0.906016] ==>[id:262031400,label:edge,weight:0.649892] //查询name为a0e05a4a-65c0-11e9-a5ce-00163e0416f8的用户的关注列表，并按照权重倒序排序。 gremlin> g.V().has('name','a0e05a4a-65c0-11e9-a5ce-00163e0416f8').outE().order().by('weight', decr).valueMap(true) //has()部分控制查询条件，order().by()部分控制排序条件。 ==>[id:252223359,label:edge,weight:0.990445] ==>[id:227652991,label:edge,weight:0.962171] ==>[id:243912806,label:edge,weight:0.906016] ==>[id:254012304,label:edge,weight:0.788503] ==>[id:217089344,label:edge,weight:0.769055] ==>[id:250193757,label:edge,weight:0.756271] ==>[id:262031400,label:edge,weight:0.649892] ==>[id:310064671,label:edge,weight:0.639453] ==>[id:316084412,label:edge,weight:0.595669] ==>[id:277559997,label:edge,weight:0.538571] ==>[id:252494754,label:edge,weight:0.494867] ==>[id:291708246,label:edge,weight:0.387777] ==>[id:281304400,label:edge,weight:0.382627] ==>[id:310008546,label:edge,weight:0.333313] ==>[id:220429042,label:edge,weight:0.290449] ==>[id:260893506,label:edge,weight:0.247677] ==>[id:300018487,label:edge,weight:0.228707] ==>[id:234881614,label:edge,weight:0.0887247] ==>[id:289510146,label:edge,weight:0.078113] ==>[id:228404583,label:edge,weight:0.0742597]
```



- 通用场景：

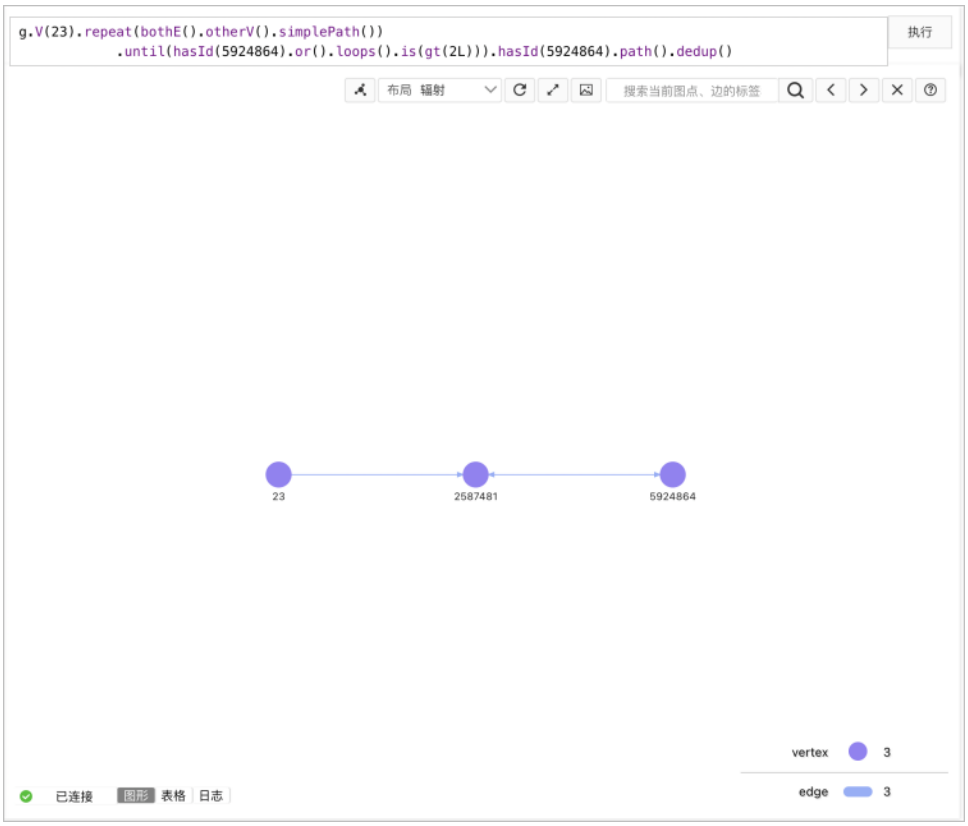
- k阶邻居：

```
//查询用户ID为23的2跳关注关系。 gremlin> g.V(23).repeat(outE('edge').otherV().simplePath()).times(2).path() //outE()部分控制查询边类型，times()部分查询深度。 ==>[v[23],e[239197952][23-edge->19201],v[19201],e[229218134][19201-edge->18246],v[18246]] ==>[v[23],e[239197952][23-edge->19201],v[19201],e[216091024][19201-edge->2586961],v[2586961]] ==>[v[23],e[267069904][23-edge->2587481],v[2587481],e[225145280][2587481-edge->1018015],v[1018015]] ==>[v[23],e[267069904][23-edge->2587481],v[2587481],e[284567757][2587481-edge->1022162],v[1022162]] ==>[v[23],e[267069904][23-edge->2587481],v[2587481],e[235313604][2587481-edge->10467758],v[10467758]] ==>[v[23],e[267069904][23-edge->2587481],v[2587481],e[265280161][2587481-edge->15350178],v[15350178]] ==>[v[23],e[267069904][23-edge->2587481],v[2587481],e[280654355][2587481-edge->1796334],v[1796334]] ......
```



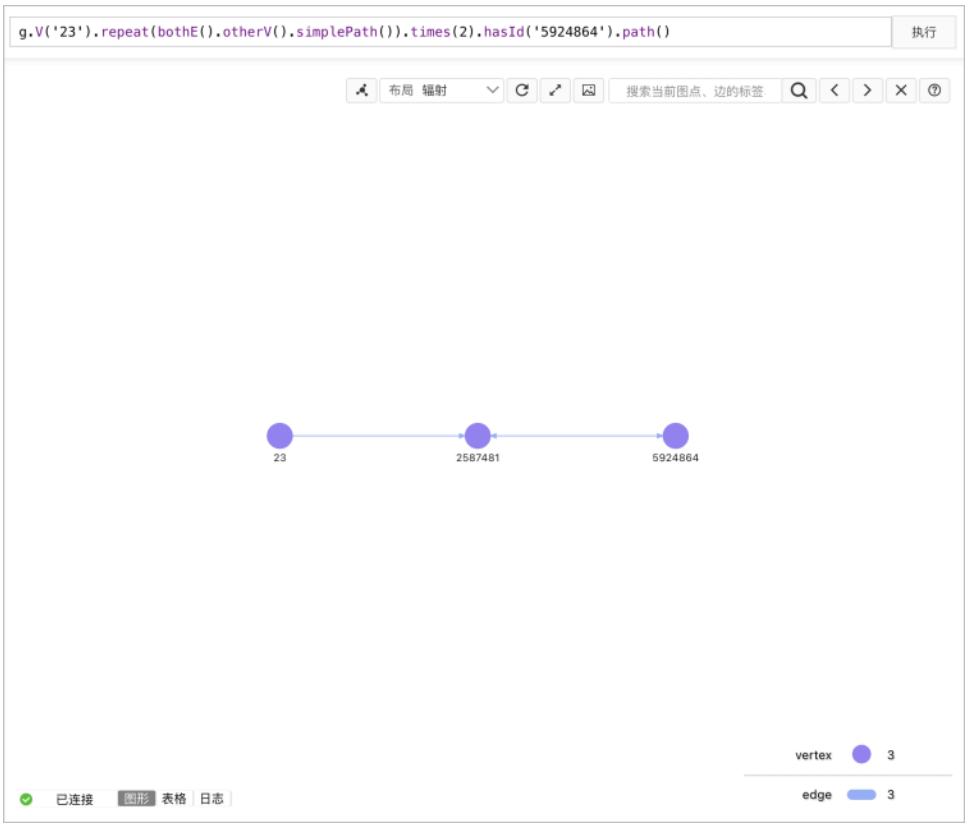
- 最短路径：

```
//查询ID为23和ID为5924864 的最短路径，最大深度为2。 gremlin> g.V(23).repeat(bothE().otherV(
).simplePath()) .until(hasId(5924864).or().loops().is(gt(2L))) //hasId()部分控制结束ID，g
t()部分查询深度。 .hasId(5924864).path().dedup() ==>[v[23],v[2587481],v[5924864]]
```



- 共同邻居：

```
//查询ID为23和ID为5924864的共同邻居。 gremlin> g.V(23).repeat(bothE().otherV().simplePath()).times(2).hasId(5924864).path().dedup() //hasId()部分控制结束ID。 ==>[v[23],e[267069904][23-edge->2587481],v[2587481],e[267070009][5924864-edge->2587481],v[5924864]]
```



- 大V查找：

```
//社交场景中，大V往往是根据粉丝的数量来衡量，粉丝越多说明越受欢迎，我们要找到拥有粉丝最多的三个大V。 gremlin> g.V().project('user','degree').by().by(inE().count()).order().by(select('degree'), desc).limit(3) // by(inE().count()) 部分控制统计逻辑，order().by(select('degree'), desc).limit(3) 控制排序逻辑 ==>[v[23],e[267069904][23-edge->2587481],v[2587481],e[267070009][5924864-edge->2587481],v[5924864]] ==>[v[23],e[267069904][23-edge->2587481],v[2587481],e[316011732][2587481-edge->5924864],v[5924864]] ==>[user:v[1],degree:1090] ==>[user:v[24],degree:890] ==>[user:v[65],degree:768]
```

- 协同推荐：

```
//推荐和ID为1的用户，有共同邻居的用户，并按照共同邻居的个数进行排序。 gremlin> g.V(1).both().aggregate("my_friend").both().has(id, neq(1)).as("ff") .flatMap(__.both().where(within("my_friend")).count()).as("comm_cnt").order().by(desc) .select("ff", "comm_cnt").dedup()
==>[ff:v[591712],comm_cnt:10] ==>[ff:v[60911],comm_cnt:10] ==>[ff:v[4470],comm_cnt:10]
==>[ff:v[47129],comm_cnt:10] ==>[ff:v[1],comm_cnt:10] ==>[ff:v[316284],comm_cnt:10] ==>
[ff:v[472652],comm_cnt:9] ==>[ff:v[52057],comm_cnt:9] ==>[ff:v[531386],comm_cnt:9] ...
```

6、 客户效果

某社交领域互联网公司，图数据规模为点（用户数据）1亿，边（社交关系）16亿，之前通过MySQL存储（32core，256GB），面临MySQL查询性能过低的问题，经常出现查询超时。通过使用图数据库GDB，以用户为节点、用户在线状态为属性，好友关系为边构建社交图谱，查询性能提升超过100倍，从超时提升到 毫秒级 。

3.保险反欺诈

在现代欺诈和各类金融犯罪中，欺诈者通过改变自身身份等达到逃避风控规则的欺诈目的。您可以通过图数据库GDB建立追踪用户行为的图结构，实时分析欺诈行为的离散数据，识别欺诈环，帮助您快速防范和解决欺诈行为。

背景信息

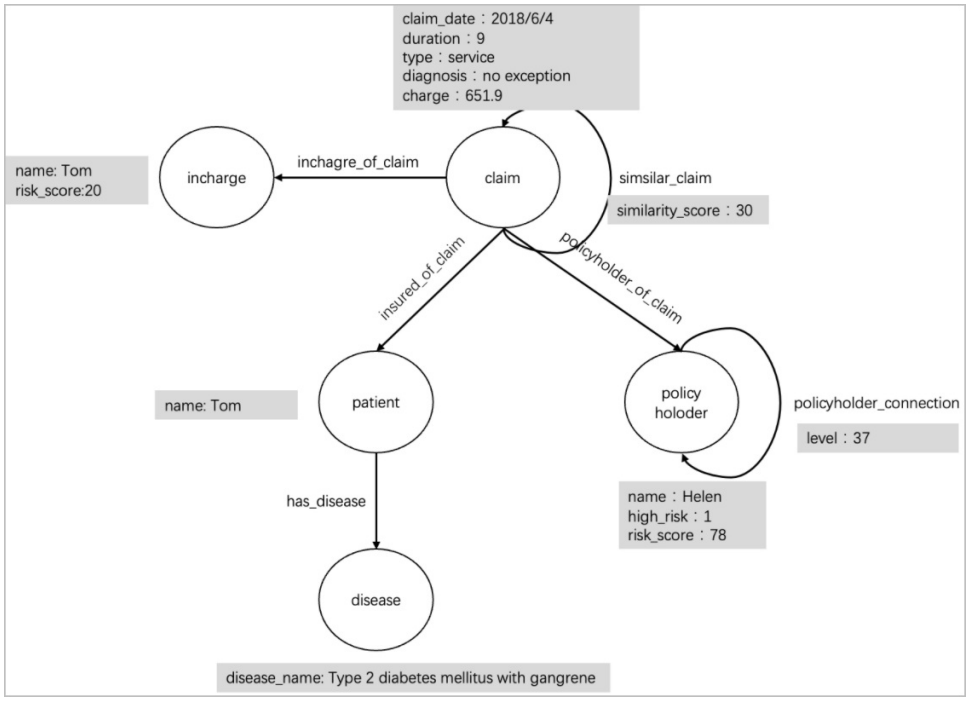
保险理赔欺诈一般根据保险提供商所具备的患者、疾病和索赔等数据，分析与被保人相关的理赔单、疾病等实体的关联关系，识别异常理赔记录，发现欺诈团伙。

1、数据和模型

以保险领域公开数据集为例。更多信息，请参见数据模型参考下载。

数据包括保险行业基本元素，可以将数据模型抽象为下图：

- 点：投保人（policyholder）、保单（incharge）、理赔（claim）、病人（patient）、疾病（disease）。
- 边：疾病-病人（has_disease）、投保人-理赔（policyholder_of_claim）、保单-理赔（inchagre_of_claim）、病人-理赔（insured_of_claim）、相似理赔（similar_claim）、投保人关联（policyholder_connection）。
- 属性：姓名（name），是否高危（high_risk）、风险分数（risk_score）、疾病名称（disease_name）、相似度（similarity_score）、关联等级（level）、理赔时间（claim_date）、保额（charge）等。



示例数据如下：

- 点文件：

```
//policyholder投保人信息。 ~id,~label,FNAME:string,LNAME:string,RISK_SCORE:int,HIGH_RISK:int PH3068,policyholder,ADAM,OCHSENBEIN,88,1 PH3069,policyholder,MALINDA,MEHSERLE,42,0 PH3070,policyholder,SANDRA,KUHTA,20,0 PH3071,policyholder,DORA,TAHU,62,1 //incharge保单信息。 ~id,~label,FNAME:string,LNAME:string,RISK_SCORE:int,SERVICE_ID:string PI18675,incharge,ROSCOE,ROSCOE,0,S10444 PI18676,incharge,AUDRIE,AUDRIE,24,S12029 PI18677,incharge,WINTER,WINTER,25,S15709 PI18678,incharge,LILA,LILA,27,S16484 PI18679,incharge,NANCIE,NANCIE,31,S12306 ...
```

- 边文件：

```
//has_disease疾病-病人。 ~id,~from,~to,~label 10529764572383,105297,64572383,has_disease 10529764572383,105297,64572383,has_disease 10529764572384,105297,64572384,has_disease 10529764572389,105297,64572389,has_disease 10941664572116,109416,64572116,has_disease 10941664572116,109416,64572116,has_disease 10941664572117,109416,64572117,has_disease //policyholder_of_claim投保人-理赔。 ~id,~from,~to,~label C1528PH2963,C1528,PH2963,policyholder_of_claim C1529PH1353,C1529,PH1353,policyholder_of_claim C1530PH1071,C1530,PH1071,policyholder_of_claim C1531PH8102,C1531,PH8102,policyholder_of_claim C1532PH4768,C1532,PH4768,policyholder_of_claim C1533PH2287,C1533,PH2287,policyholder_of_claim C1534PH6948,C1534,PH6948,policyholder_of_claim C1535PH621,C1535,PH621,policyholder_of_claim
```

2、创建实例

创建图数据库实例。具体操作，请参见创建主实例。

> **说明**
>
> - 实例购买成功后，您可以在**实例列表**页面查看实例相关信息。通常，实例创建成功需要3~5分钟。
> - 创建GDB实例后，您需要创建账号和密码、设置安全组，保证您具有GDB实例的访问权限。具体操作，请参见创建账号和设置白名单。

3、数据导入

图数据库GDB支持从多种数据源将数据导入至图数据库GDB，您可以使用以下两种方式进行数据导入：

- 从阿里云对象存储OSS导入数据至图数据库GDB实例。
- 使用DataWorks导入数据至图数据库GDB实例。

4、连接实例

图数据库GDB支持多种方法连接实例，您可以通过以下五种方式连接实例：

- 通过GDB控制台直接登录数据管理服务DMS，更加方便快捷地远程访问、在线管理您的GDB数据库。具体操作，请参见通 过DMS登录GDB数据库。
- 通过开源组件GDB Console可视化控制台登录图数据库，可视化界面，操作简单，并可根据业务需求对可视化界面进行二次开发。具体操作，请参见通过开源组件GDB Console登录图数据库。
- 通过Gremlin Console连接实例，命令行模式，适合Gremlin内核版本，适合用于查询语句性能优化。具体操作，请参见通过Gremlin Console连接实例。
- 通过Cypher Shell连接实例，命令行模式，适合Cypher内核版本。具体操作，请参见通过Cypher Shell连接实例 。
- 通过SDK连接，支持Java、Python、.Net、Go、Node.js五种SDK。具体操作，请参见SDK参考。

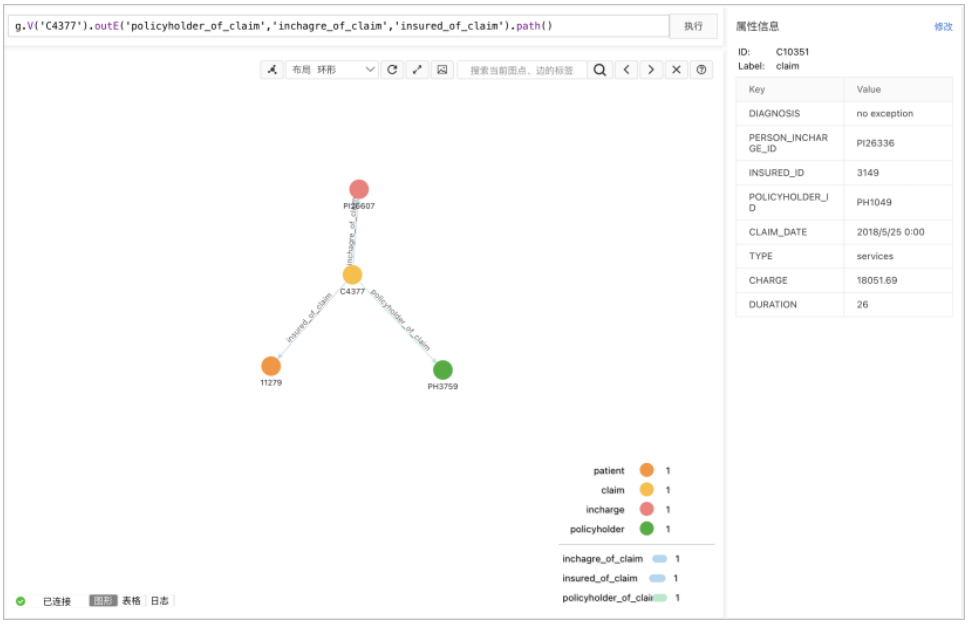
5、使用范例

- 简单查询
  - 数据统计：

```
//统计点的数目。 gremlin> g.V().count() ==>120571 //统计边的数目。 gremlin> g.E().count() ==>215943 //统计每种类型点的数量。 gremlin> g.V().group().by(label).by(count()) ==>[policyholder:10006,disease:397,patient:166,claim:100001,incharge:10001] //统计每种类型边的数量。 gremlin> g.E().group().by(label).by(count()) ==>[inchagre_of_claim:100001,insured_of_claim:12,similar_claim:15279,policyholder_connection:347,policyholder_of_claim:99983,has_disease:321]
```

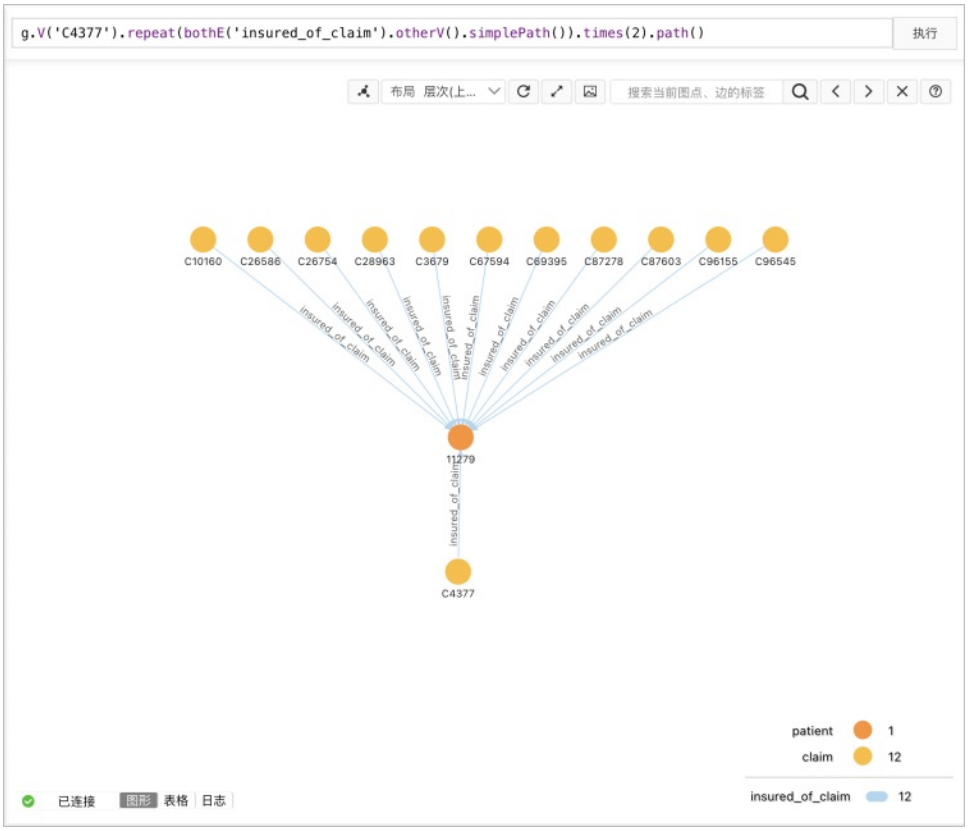
  - 过滤查询、排序查询：

```
//查询理赔单C4377的投保、理赔、被保情况。 gremlin> g.V('C4377').outE('policyholder_of_claim','inchagre_of_claim','insured_of_claim').path() ==>[v[C4377],e[C4377PI26607][C4377-inchagre_of_claim->PI26607]] ==>[v[C4377],e[C437711279][C4377-insured_of_claim->11279]] ==>[v[C4377],e[C4377PH3759][C4377-policyholder_of_claim->PH3759]]
```

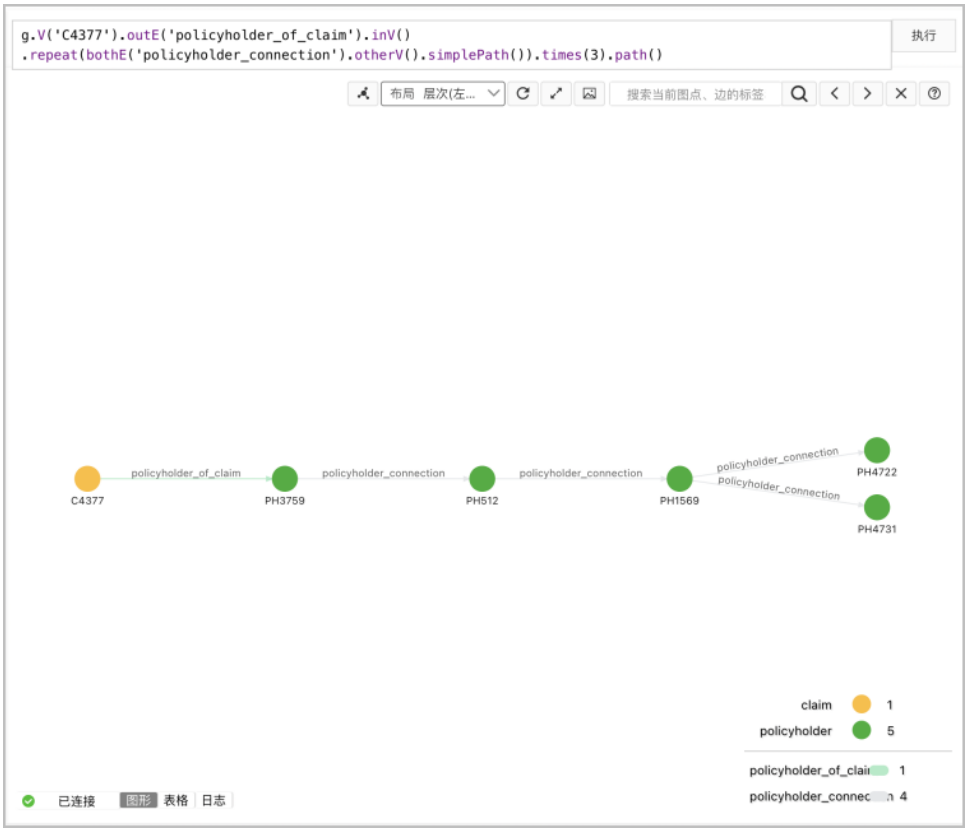


- 通用场景
  - K阶邻居：

//已知保单C4377为欺诈保单，查询和理赔单C4377有相同理赔病人的理赔单，说明该理赔人有涉嫌骗保的嫌疑。
gremlin> g.V('C4377').repeat(bothE('insured_of_claim').otherV().simplePath()).times(2).path() ==>[v[C4377],e[C437711279][C4377-insured_of_claim->11279],v[11279],e[C1016011279][C10160-insured_of_claim->11279],v[C10160]] ==>[v[C4377],e[C437711279][C4377-insured_of_claim->11279],v[11279],e[C2658611279][C26586-insured_of_claim->11279],v[C26586]] ==>[v[C4377],e[C437711279][C4377-insured_of_claim->11279],v[11279],e[C2675411279][C26754-insured_of_claim->11279],v[C26754]] ==>[v[C4377],e[C437711279][C4377-insured_of_claim->11279],v[11279],e[C2896311279][C28963-insured_of_claim->11279],v[C28963]] ==>[v[C4377],e[C437711279][C4377-insured_of_claim->11279],v[11279],e[C367911279][C3679-insured_of_claim->11279],v[C3679]] ==>[v[C4377],e[C437711279][C4377-insured_of_claim->11279],v[11279],e[C6759411279][C67594-insured_of_claim->11279],v[C67594]] ==>[v[C4377],e[C437711279][C4377-insured_of_claim->11279],v[11279],e[C6939511279][C69395-insured_of_claim->11279],v[C69395]] ==>[v[C4377],e[C437711279][C4377-insured_of_claim->11279],v[11279],e[C8727811279][C87278-insured_of_claim->11279],v[C87278]] ==>[v[C4377],e[C437711279][C4377-insured_of_claim->11279],v[11279],e[C8760311279][C87603-insured_of_claim->11279],v[C87603]] ==>[v[C4377],e[C437711279][C4377-insured_of_claim->11279],v[11279],e[C9615511279][C96155-insured_of_claim->11279],v[C96155]] ==>[v[C4377],e[C437711279][C4377-insured_of_claim->11279],v[11279],e[C9654511279][C96545-insured_of_claim->11279],v[C96545]]

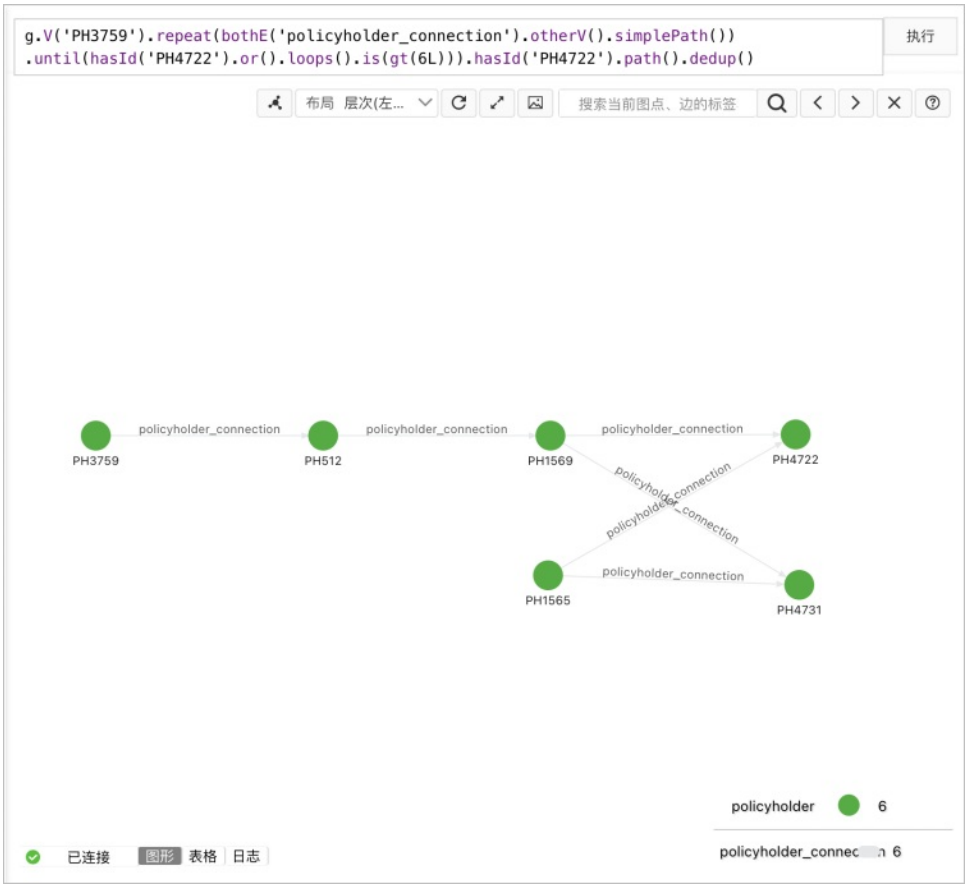


//查询已知欺诈保单C4377的投保人的社交关系，可以对这些人的理赔情况提前预警。 gremlin> g.V('C4377').outE('policyholder_of_claim').inV().repeat(bothE('policyholder_connection').otherV().simplePath()).times(3).path() ==>[v[C4377],e[C4377PH3759][C4377-policyholder_of_claim->PH3759],v[PH3759],e[PH3759PH512][PH3759-policyholder_connection->PH512],v[PH512],e[PH512PH1569][PH512-policyholder_connection->PH1569],v[PH1569],e[PH1569PH4722][PH1569-policyholder_connection->PH4722],v[PH4722]] ==>[v[C4377],e[C4377PH3759][C4377-policyholder_of_claim->PH3759],v[PH3759],e[PH3759PH512][PH3759-policyholder_connection->PH512],v[PH512],e[PH512PH1569][PH512-policyholder_connection->PH1569],v[PH1569],e[PH1569PH4731][PH1569-policyholder_connection->PH4731],v[PH4731]]



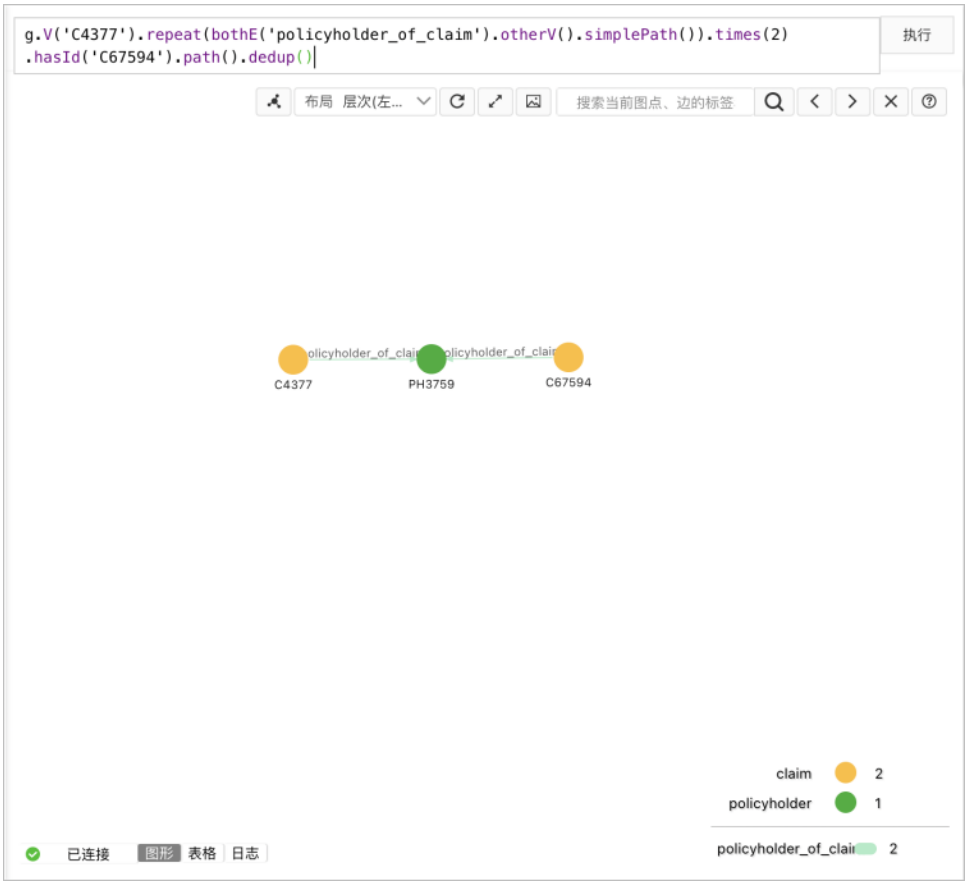
- 最短路径：

```
//查询投保人投保人PH3759和投保人PH4722 的最短路径，分析投保人之间的关联关系。 gremlin> g.V('PH3759').repeat(bothE().otherV().simplePath()) .until(hasId('PH4722').or().loops().is(gt(3L))) //hasId()部分控制结束ID，gt()部分查询深度。 .hasId('PH4722').path().dedup() ==>[v[PH3759],e[PH3759PH512][PH3759-policyholder_connection->PH512],v[PH512],e[PH512PH1569][PH512-policyholder_connection->PH1569],v[PH1569],e[PH1569PH4722][PH1569-policyholder_connection->PH4722],v[PH4722]] ==>[v[PH3759],e[PH3759PH512][PH3759-policyholder_connection->PH512],v[PH512],e[PH512PH1569][PH512-policyholder_connection->PH1569],v[PH1569],e[PH1569PH4731][PH1569-policyholder_connection->PH4731],v[PH4731],e[PH1565PH4731][PH1565-policyholder_connection->PH4731],v[PH1565],e[PH1565PH4722][PH1565-policyholder_connection->PH4722],v[PH4722]]
```


g.V('PH3759').repeat(bothE('policyholder_connection').otherV().simplePath())
.until(hasId('PH4722').or().loops().is(gt(6L))).hasId('PH4722').path().dedup()
执行
布局 层次(左...
搜索当前图点、边的标签
policyholder_connection
PH3759
PH512
PH1569
PH4722
PH1565
PH4731
policyholder 6
已连接
图形 表格 日志

- 共同邻居：

```
//查询保单C4377和保单C67594的共同邻居，从而找到两个保单的共同投保人。 gremlin> g.V('C4377').repeat(bothE('policyholder_of_claim').otherV().simplePath()).times(2).hasId('C67594').path().dedup() //hasId()部分控制结束ID。 ==>[v[C4377],e[C4377PH3759][C4377-policyholder_of_claim->PH3759],v[PH3759],e[C67594PH3759][C67594-policyholder_of_claim->PH3759],v[C67594]]==>[v[2987000],e[3][2987000-t_a1->315.0],v[315.0],e[938][2987172-t_a1->315.0],v[2987172]] ==>[v[2987000],e[4][2987000-t_a2->87.0],v[87.0],e[939][2987172-t_a2->87.0],v[2987172]] ==>[v[2987000],e[5][2987000-t_p->W],v[W],e[940][2987172-t_p->W],v[2987172]]
```



- 协同推荐：

```
//已知保单C4377为欺诈保单，查找和保单C4377有共同投保人的保单，并按照共同邻居的个数进行排序，从而找到欺诈疑似涉诈保单。 gremlin> g.V('C4377').both('policyholder_of_claim').aggregate("policyholder_of_claim").both('policyholder_of_claim').has(id, neq('C4377')).as("recomm_claim") .flatMap(__.both().where(within("policyholder_of_claim")).count()).as("comm_cnt").order().by(desc) .select("recomm_claim", "comm_cnt").dedup() ==>[recomm_claim:v[C26754],comm_cnt:1] ==>[recomm_claim:v[C26586],comm_cnt:1] ==>[recomm_claim:v[C10160],comm_cnt:1] ==>[recomm_claim:v[C28963],comm_cnt:1] ==>[recomm_claim:v[C3679],comm_cnt:1] ==>[recomm_claim:v[C67594],comm_cnt:1] ==>[recomm_claim:v[C69395],comm_cnt:1] ==>[recomm_claim:v[C87278],comm_cnt:1] ==>[recomm_claim:v[C87603],comm_cnt:1] ==>[recomm_claim:v[C96155],comm_cnt:1] ==>[recomm_claim:v[C96545],comm_cnt:1]
```

- 与已知涉诈保单相似的保单：

```
//查询和相似的保单，根据相似度分数取前10。 gremlin> g.V('C4377') .sideEffect(out().store('v1
n')) .as('v1') .select('v1n').unfold().in().limit(100).simplePath().dedup().as('v2') .p
roject('i', 'u') .by(select('v2').out().where(within('v1n')).count()) .by(union(select(
'v2').out().fold(),select('v1n')).unfold().dedup().count()) .project('Transaction', '相
似度') .by(select('v2').id()) .by(math('i/u')) .order().by(select('相似度'), desc).limit
(10) ==>[Transaction:C96545,相似度:0.992619926199262] ==>[Transaction:C15383,相似度:0.98
71086556169429] ==>[Transaction:C17300,相似度:0.9871086556169429] ==>[Transaction:C30519
,相似度:0.9871086556169429] ==>[Transaction:C56230,相似度:0.9871086556169429] ==>[Transa
ction:C60174,相似度:0.9871086556169429] ==>[Transaction:C6591,相似度:0.9871086556169429]
==>[Transaction:C66712,相似度:0.9871086556169429] ==>[Transaction:C68975,相似度:0.987108
6556169429] ==>[Transaction:C7181,相似度:0.9871086556169429]
```

6、客户效果

某头部保险公司，经测算其关联查询性能是原有保险反欺诈方案的10~100倍。在原先的客户信息系统中，查询一个保单关联的信息（投保、理赔、疾病等信息），是通过两表之间的多次JOIN实现，当属性多、逻辑复杂时需要经过几十次甚至百次数据库交互，网络通信的开销带来极大的性能损耗。通过使用阿里云图数据库GDB的存储方案，可以带来10~100倍的性能提升。

4.知识图谱场景

知识图谱是由Google公司在2012年提出来的一个新的概念。从学术的角度，我们可以对知识图谱给一个这样的定义：“知识图谱本质上是语义网络（Semantic Network）的知识库”。但这有点抽象，所以换个角度，从实际应用的角度出发其实可以简单地把知识图谱理解成多关系图（Multi-relational Graph）。

我们可以通过一定方法把知识抽取和清洗出来，然后存入GDB中提供查询。搜索引擎只能告诉用户，查询的结果与哪些页面相关用户需要肉眼在页面里找答案，知识图谱可以直接把答案告诉用户。比如通过知识图谱我们能直接告诉用户在《权力的游戏》中坦格利安家族的伊利亚丈夫的兄妹是谁。

应用举例

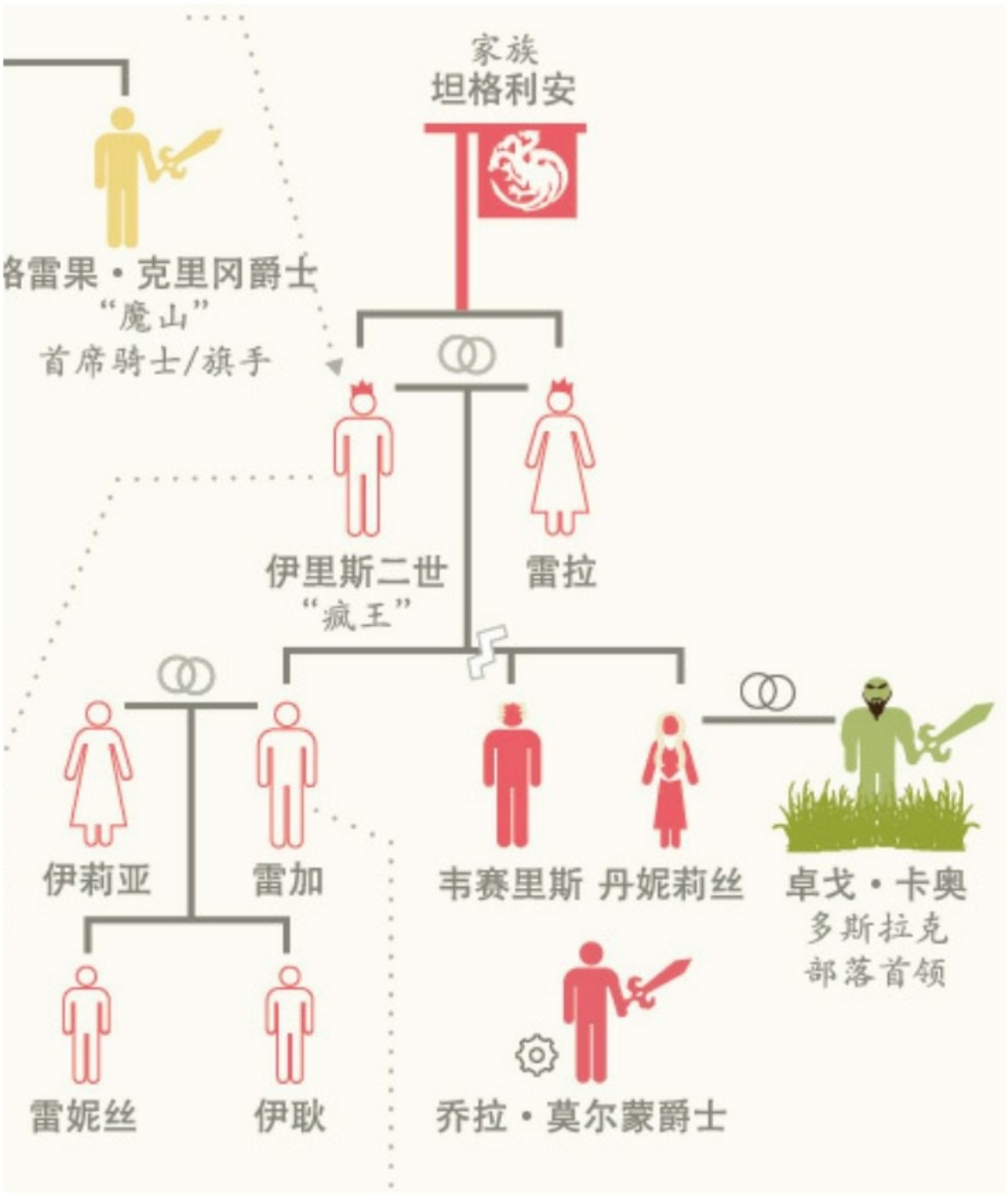
假如我们现在有了《权力的游戏》里各个人物的信息和人物的相互关系（以坦格利安家族为例），如：

subject	predicate	object
伊里斯二世“疯王”	妻子	雷拉
雷加	兄妹	韦赛里斯
伊莉亚	丈夫	雷加
雷加	兄妹	丹妮莉丝
雷加	儿子	雷妮丝
雷加	儿子	伊耿
丹妮莉丝	丈夫	卓戈-卡奥
伊里斯二世	子女	雷加

阿里云图数据库服务GDB目前支持的图模型是属性图，属性图将存储的基本元素分为点（Vertex）、边（Edge）和属性（Property），每个Vertex或Edge都有一个标签（Label）来代表它的类型。知识图谱三元组我们就分别以Vertex存subject和object，Edge存predicate。

举例

这里假设我们已经抽取到了权游的人物关系，并且存到了GDB中。



雷加的兄妹叫什么名字

```
g.V().has("name", "雷加").out("兄妹").values("name")
```

- `g.V().has(“name”, “雷加”)`：找到GDB中雷加这个vertex。
- `out("兄妹").values("names")`：把雷加这个顶点的所有出边是兄妹的终点找到并输出名字。

伊莉亚丈夫的父亲叫什么名字

```
g.V().has("name", "伊莉亚").in("妻子").in("子女").values("name")
```

- `g.V().has("name", "伊莉亚").in("妻子")`：找到伊莉亚的丈夫雷加。
- `in("子女").values("name")`：找到雷加的双亲。

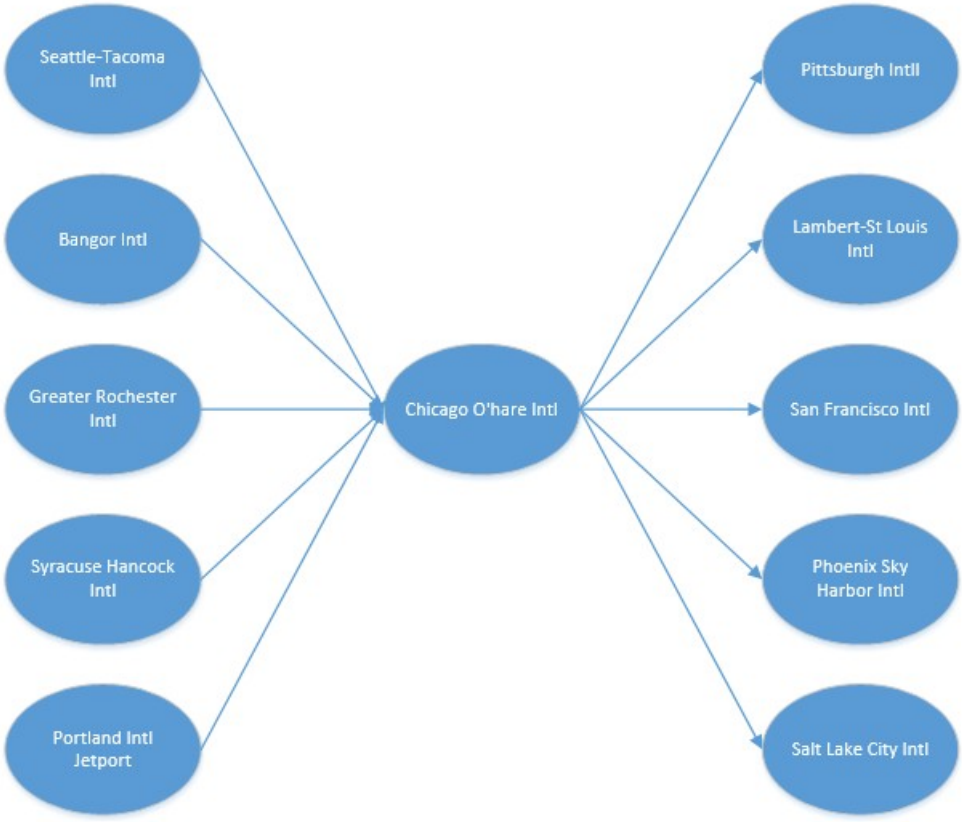
5.网络运维场景

图数据库GDB可以轻松的对IT、网络运营的海量数据进行建模，将原始关系型数据库中结构化数据转换图数据库中的节点和边，不仅能减少DBA的数据建模工作量，又能将提升查询效率。解决传统解决方案中数据量超过一定量级时不能使用数据库和使用CMDB周期长且不够灵活的问题。

下面示例采用USAir97的机场运输的dataset对具体应用加以说明。

示例示意图

航站枢纽节点连接示意图如下：



示例数据模型

图数据模型抽象以每个机场航运站为节点，航运站间的运输路线为边。其中：

- 节点中包含航运站名称，航运站坐标位置。
- 边中包含所连接的航运站，及航运站之间的距离。

代码示例

- 查看一共有多少点。

```
g.V().count()
```

- 查看一共有多少边。

```
g.E().count()
```

- 查看枢纽节点，一共有多少节点连入。

```
g.V('118').in().count() //118为"Chicago O'hare Intl"节点
```

一共有哪些连入节点。

```
g.V('118').in().id()
```

返回结果：

```
65  //"Portland Intl"
94  //"General Mitchell Intll"
95  //"Greater Buffalo Intl"
8   //"Anchorage Intl"
...
```

- 查看枢纽节点，一共连了多少节点。

```
g.V('118').out().count()
```

一共连了哪些节点。

```
g.V('118').out().id()
```

返回结果：

```
201  //"San Francisco Intl"
221  //"Raleigh-Durham Intll"
301  //"Tampa Intl"
232   //"Memphis Intl"
...
```

- 最优运输路径选择： 从” Wiley Post-Will Rogers Mem” 节点运输货物至 “Shreveport Regional” 的最优路径选择。

```
g.V("1").store("x").repeat(out().where(without("x")).aggregate("x")).until(hasId("267")).
path()
```

返回结果：

```
[v[1],v[4],v[47], v[255],v[267]]
// "Wiley Post-Will Rogers Mem" -> "Fairbanks Intl" -> "Seattle-Tacoma Intl" -> "The Will
iam B Hartsfield Atlan" -> "Shreveport Regional"
```

- 运输路线统计分析：每次运输一个节点添加运输货物事件记录，查询时可按货物查询运输路线，达到运输路线统计，货物追踪及货物信息统计等。

```
g.V('1').property('cargo1', 1)
g.V().properties('cargo1').valueMap(true)
```

返回结果：

```
v[1]
[id:1,key:cargo1,value:1]
[id:4,key:cargo1,value:2]
[id:47,key:cargo1,value:3]
```

- 货物运输路线跟踪：在货物运输完或运输途中，可以根据图数据库中留下的记号，追踪货物在运输网中的路径。

```
g.V().has("cargo1").valueMap(true)
```

返回结果：

```
[id:1,label:vertex,cargo1:[1]]
[id:4,label:vertex,cargo1:[2]]
[id:47,label:vertex,cargo1:[3]]
```

6.推荐场景

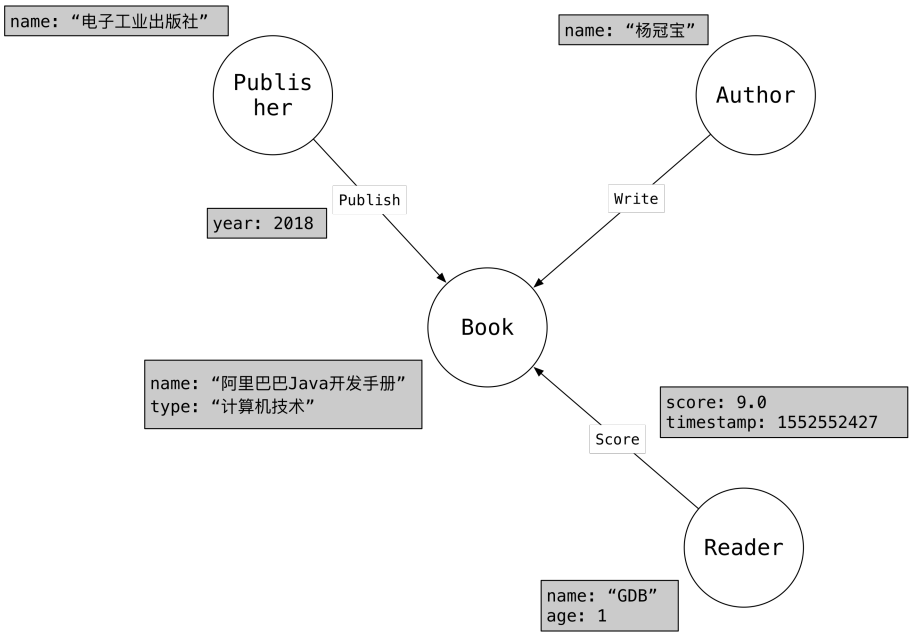
手机淘宝购物时，手淘上有**猜你喜欢**页面来推荐你可能会感兴趣的商品；使用优酷观看电影时，页面上也会推荐你可能喜欢的电影列表。个性化推荐的应用非常广泛，帮助企业挖掘潜在的用户需求、提高转化率的同时，也能够有效提升用户体验度。个性化推荐基于用户的历史行为习惯，商品、电影本身的属性等数据集，这些数据之间相互关联，应用推荐策略时需要利用这些数据之间复杂的关联关系，因此使用图数据库来存储这些数据是非常合适的。

建模

本文以书籍的个性化推荐举例，来说明如何使用图数据库的个性化推荐应用。

图数据库GDB目前支持的图模型是属性图，属性图将存储的基本元素分为点（Vertex）、边（Edge）和属性（Property），每个Vertex或Edge都有一个标签（Label）来代表它的类型。在书籍个性化推荐中，我们可以定义如下的基本元素：

- Vertex：书籍（Book）、出版商（Publisher）、作者（Author）、读者（Reader）。
- Edge：出版（Publish）、写作（Write）、打分（Score）。
- Property：名字（name）、类型（type）、年龄（age）、出版年份（year）。



代码实现

现在请您给用户名为“小明”的读者推荐书籍，直观想法是“找到那些和小明评价过相同的书的读者，把他们评价过的其它书推荐给小明”。

图数据库GDB目前支持的查询语言是Gremlin，下面使用Gremlin语句来实现：

1. 找到小明。

```
g.V().hasLabel('Reader').has('name','小明')
```

2. 找到小明评价过的书。

```
g.V().hasLabel('Reader').has('name','小明').as('myself').out('Score')
```

3. 找到跟小明评价过相同书籍的其他人。

```
g.V().hasLabel('Reader').has('name','小明').as('myself').out('Score').aggregate('scored_
books').in('Score').where(neq('myself'))
```

4. 找到这些人看过的小明没有看过的书。

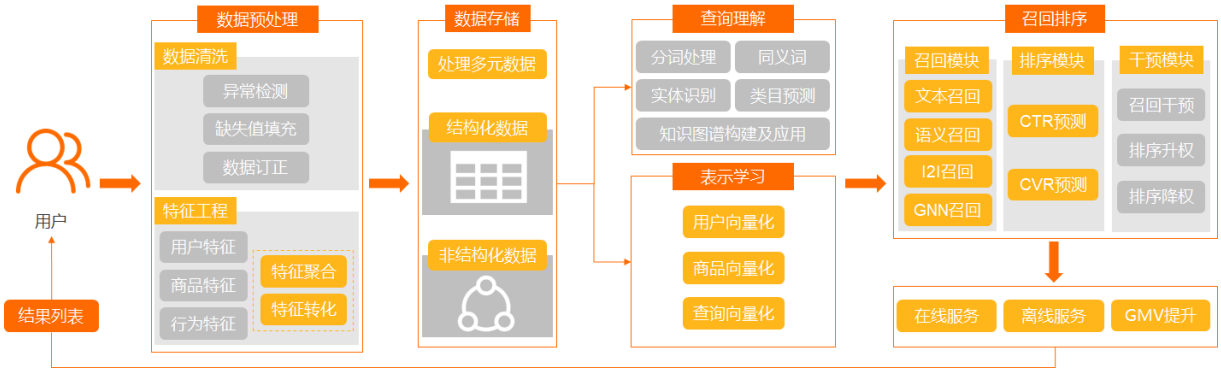
```
g.V().hasLabel('Reader').has('name','小明').as('myself').out('Score').aggregate('scored_
books').in('Score').where(neq('myself')).out('Score').where(not(within('scored_books'))
)
```

7.智能搜索推荐场景

7.1. 智能搜索推荐一体化营收增长解决方案

图数据库GDB提供智能搜索推荐一站式服务，基于达摩院的智能搜索推荐算法和知识图谱技术，助力企业快速过渡冷启动过程，面向业务场景定制化方案，以提升核心业务指标，实现业务营收增长。

方案架构



方案特点

- 智能算法解决数据偏差和兴趣迁移挑战

  通过智能算法解决由于商品曝光、用户的选择偏差和从众心理等因素造成的数据缺失、数据偏差问题，同时解决因时序变化而使用户兴趣发生变化等问题，向用户推荐用户想要的商品。

- 多种策略召回深度挖掘用户偏好

  基于多种算法策略捕捉用户、商品和行为特征，结合元路径和图神经网络算法实现千人千面的个性化推荐，并具备可解释性。

- 知识图谱助力营销收入增长

  借助知识图谱技术，精准理解用户和商品之间的交互行为，通过交互行为产生的信息，找到可运营因素，帮助业务营销收入增长。

- 开放接口支持定制化业务场景需求

  提供开放接口支持企业根据业务场景定制个性化方案，例如设置白名单或黑名单，可用于大型促销等场景。

方案优势

- 提供搜索推荐一站式服务

  将搜索和推荐融合提供一站式服务，精准理解用户意图并提供个性化智能推荐，推用户之所想。

- 融合知识图谱技术和数据沉淀

  运用知识图谱技术，融合阿里电商丰富策略和数据沉淀，提供行业知识驱动、多元场景覆盖的智能服务。

- 切实提升核心业务营收指标

  面向用户业务场景，通过智能算法和策略的赋能，同时支持用户面向一些业务场景的定制化需求，切实提升营收指标。

可解决问题

- 优化搜索匹配效果。
- 千人千面精准分发和推荐。
- 新商品和用户快速冷启动。
- 支持方案场景化定制。

应用场景

电商、新零售或泛推荐：精准理解用户搜索意图，个性化千人千面推荐分发，切实提升GMV营收指标。

方案实例

某奢侈品电商将智能搜索推荐服务接入自己的核心搜索业务，智能搜索推荐服务帮助客户优化搜索推荐结果，打造出“猜您喜欢”的个性化推荐方案，使该电商的新用户平均GMV（Gross Merchandise Volume）提升61.88%，平均点击率提升18.44%。

申请方法

智能搜索推荐一体化营收增长方案的试用活动已上线，您可以登录（试用申请）智能搜索推荐一体化营收增长方案申请试用（不支持RAM账号试用），具体请参见申请智能搜索推荐解决方案服务。

7.2. 申请智能搜索推荐解决方案服务

图数据库GDB提供智能搜索推荐一站式解决方案，面向您的具体业务场景提供服务，以提升业务指标，实现营销收入增长。本文介绍申请智能搜索推荐解决方案的方法。

前提条件

已注册阿里云账号，注册方法请参见 注册阿里云账号 。

操作步骤

1. 登录智能搜索推荐解决方案。
2. 在**智能搜索推荐解决方案**页面，设置以下参数。

参数	参数说明
阿里云账号	默认为您的阿里云账号。
公司名称	输入公司名称。
手机号码	输入与钉钉绑定的手机号码。 **说明** 您可以通过该手机号码接收审核结果。
公司预算	输入计划未来每年的预算。
所在行业	输入公司所属行业。
业务场景	输入产品的使用场景。

3. 单击**提交**。