##### Neo4j概述

1. **介绍**

neo4j是目前排名最高的图数据库,分为商业和社区版本,社区版只支持单机,而且查询的运行时(runtime)不同(cypher runtime:interpreted(社区版),slotted(企业版))。 Neo4j是目前应用最为广泛的图数据库，它特别适合于处理复杂网络关系。Neo4j是一个高性能的NoSQL图形数据库，它是目前对图计算支持的最好的数据库（没有之一），底层用JAVA开发，支持完整的事务处理。Neo4j使用的查询语言是Cypher，该语言能够实现类似于标准SQL的复杂查询。

neo4j最初的设计动机是为了更好的描述实体之间的联系，现实生活中，每个实体都与周围的其他实体有着千丝万缕的关系，这些关系中存在着大量的潜在信息，但是传统的关系型数据库更加注重刻画实体内部的属性，实体与实体之间的关系往往需要通过外键来实现，因此，在查询一个实体的关系时需要join操作，特别是深层次的关系查询需要大量的join操作，而join操作通常又非常的耗时，特别是深层次的关系查询时需要大量的join操作，而join操作通常又非常的耗时，所以为了应对海量数据查询深层关系时的性能问题、以及运算负责性，neo4j应运而生。

neo4j数据库中只有3个概念: Node, Relationship, Properties. Node表示实体类别,使用Label区分,例如一个节点可以有Person/Father等多个标签,Relationship即关系,雇佣关系,父子关系,投资关系,交易关系等. Node和Relationship都可以有Proerties,属性自身不分是属于节点还是属于关系,例如Person可以有属性name,关系也可以用属性name.你可以在neo4j browser左侧看到当前数据库的所有Node Label,Relationship Type,Properties。

Neo4j分为社区版（免费）和企业版（收费），社区版和企业版主要区别在如下几点：

* 容量：社区版最多支持 320 亿个节点、320 亿个关系和 640 亿个属性，而企业版没有限制；
* 部署：社区版只能部署成单实例，不支持集群部署，而企业版可以部署成高可用集群；
* 性能：社区版最多使用 4 个内核，而企业版没有限制，而且对性能做了优化；
* 支持：享受服务，企业版客户能得到技术支持服务。

**二、neo4j的体系结构**

**1）免索引邻接**

neo4j有一个重要的特点，就是用来保证关系查询的速度，即免索引邻接属性，数据库中的每个节点都会维护与它相邻节点的引用。因此每个节点都相当于与它相邻节点的微索引，这比使用全局索引的代价小很多，这就意味着查询时间和图的整体规模无关，只与它附近节点的数量成正比。在关系数据库中使用全局索引引接各个节点，这些索引对每个遍历都会增加一个中间层，因此会导致非常大的计算成本，而免索引邻接为图数据库提供了快速、高效的图遍历能力。

**2）neo4j底层存储结构**

免索引邻接是图数据实现高效遍历的关键，那么免索引邻接的实现机制就是neo4j底层存储结构设计的关键。能够支持高效的、本地化的图存储以及支持任意图算法的快速遍历，是使用图数据库的重要原因。

从宏观角度讲，neo4j只存在两种数据类型：

* 节点：节点类似于ER图中的实体，每一个实体可以有零个或多个属性，这些属性以key-value对的形式存在，属性没有特殊类别的要求，同时每个节点还具有相应的标签，用来区分不同类型的节点。
* 关系：关系也类似ER图中的关系，一个关系有起始节点和终止节点，另外，关系也能够有自己的属性和标签。

##### Neo4j功能和特点

1. **Neo4j的特点：**

* 它拥有简单的查询语言 Neo4j CQL
* 它遵循属性图数据模型
* 它通过使用 Apache Lucence 支持索引
* 它支持 UNIQUE 约束
* 它包含一个用于执行 CQL 命令的 UI：Neo4j 数据浏览器
* 它支持完整的 ACID（原子性，一致性，隔离性和持久性）规则
* 它采用原生图形库与本地 GPE（图形处理引擎）
* 它支持查询的数据导出到 Json 和 XLS 格式
* 它提供了 REST API，可以被任何编程语言（如 Java，Spring，Scala 等）访问
* 它提供了可以通过任何 UI MVC 框架（如 Node JS ）访问的 Java 脚本
* 它支持两种 Java API：Cypher API 和 Native Java API 来开发 Java 应用程序

1. **neo4j 基本操作元素**

neo4j可支持语言：.NET、Java、Spring、JavaScript、Python、Ruby、PHP、R、Go、C / C++、Clojure、Perl、Haskell

##### Neo4j相关产品对比

**一、简介**

在众多不同的数据模型里，关系数据模型自20世纪80年代就处于统治地位，而且出现了不少巨头，如Oracle、MySQL和MSSQL，它们也被称为关系数据库管理系统（RDBMS）。然而，随着关系数据库使用范围的不断扩大，也暴露出一些它始终无法解决问题，其中最主要的是数据建模中的一些缺陷和问题，以及在大数据量和多服务器之上进行水平伸缩的限制。同时，互联网发展也产生了一些新的趋势变化：

* 用户、系统和传感器产生的数据量呈指数增长，其增长速度因大部分数据量集中在Amazon、Google和其他云服务的分布式系统上而进一步加快；
* 数据内部依赖和复杂度的增加，这一问题因互联网、Web2.0、社交网络，以及对大量不同系统的数据源开放和标准化的访问而加剧。

而在应对这些趋势时，关系数据库产生了更多的不适应性，从而导致大量解决这些问题中某些特定方面的不同技术出现，它们可以与现有RDBMS相互配合或代替它们——亦被称为混合持久化（Polyglot Persistence）。数据库替代品并不是新鲜事物，它们已经以对象数据库（OODBMS）、层次数据库（如LDAP）等形式存在很长时间了。但是，过去几年间，出现了大量新项目，它们被统称为NoSQL数据库（NoSQL-databases）。

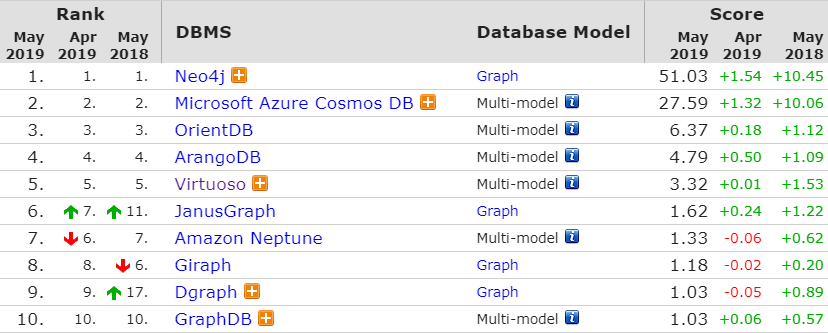
**二、NoSQL数据库**

NoSQL（Not Only SQL，不限于SQL）是一类范围非常广泛的持久化解决方案，它们不遵循关系数据库模型，也不使用SQL作为查询语言。其数据存储可以不需要固定的表格模式，也经常会避免使用SQL的JOIN操作，一般有水平可扩展的特征。

简言之，NoSQL数据库可以按照它们的数据模型分成4类：

* 键-值存储库（Key-Value-stores）
* BigTable实现（BigTable-implementations）
* 文档库（Document-stores）
* 图形数据库（Graph Database）

在NoSQL四种分类中，图数据库从最近十年的表现来看已经成为关注度最高，而neo4j是发展趋势最明显的图数据库。



**三、图形数据库和关系数据库分析对比**

NoSQL（Not Only SQL）数据库泛指非关系型的数据库。数据库常常可以用来处理传统的关系型数据库所难以解决的一系列问题。通常情况下，这些NoSQL数据库分为Graph，Document，Column Family以及Key-Value Store等四种，这四种类型的数据库分别使用了不同的数据结构来记录数据，所适用的场景也不尽相同。

其中图形数据库和其它的NoSQL数据库非常不同：丰富的关系表示，完整的事务支持，却没有一个纯正的横向扩展解决方案。

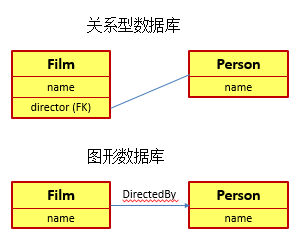
    在使用关系型数据库时常常会遇到一系列非常复杂的设计问题。这种彼此关联的关系常常会非常复杂，而且在两个实体之间常常同时存在着多个不同的关系。在尝试使用关系型数据库对这些关系进行建模时，我们首先需要建立表示各种实体的一系列表，这些表常常需要通过一系列关联表将它们关联起来，我们需要大量的关联表来记录这一系列复杂的关系。在更多实体引入之后，我们将需要越来越多的关联表，从而使得基于关系型数据库的解决方案繁琐易错。

    这主要是因为关系型数据库是以为实体建模这一基础理念设计的，并没有提供对这些实体间关系的直接支持。需要创建一个关联表以记录这些数据之间的关联关系，而且关联表常常不用来记录除外键之外的其它数据，也就是说这些关联表也仅仅是通过关系型数据库所已有的功能来模拟实体之间的关系。这种模拟导致了两个非常糟糕的结果：数据库需要通过关联表间接地维护实体间的关系，导致数据库的执行效能低下;同时关联表的数量急剧上升。

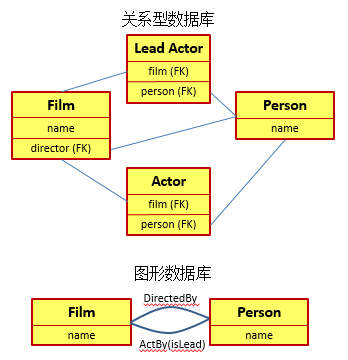
在需要描述大量关系时，传统的关系型数据库已经不堪重负。它所能承担的是较多实体但是实体间关系略显简单的情况。而对于这种实体间关系非常复杂，常常需要在关系之中记录数据，而且大部分对数据的操作都与关系有关的情况，原生支持了关系的图形数据库才是正确的选择。它不仅仅可以为我们带来运行性能的提升，更可以大大提高系统开发效率，减少维护成本。

在一个图形数据库中，数据库的最主要组成主要有两种，结点集和连接结点的关系。结点集就是图中一系列结点的集合，比较接近于关系数据库中所最常使用的表。而关系则是图形数据库所特有的组成。因此对于一个习惯于使用关系型数据库开发的人而言，如何正确地理解关系则是正确使用图形数据库的关键。

图形数据库对数据进行抽象的方式实际上和关系型数据库非常接近。每个结点仍具有标示自己所属实体类型的标签，也既是其所属的结点集，并记录一系列描述该结点特性的属性。除此之外，我们还可以通过关系来连接各个结点。

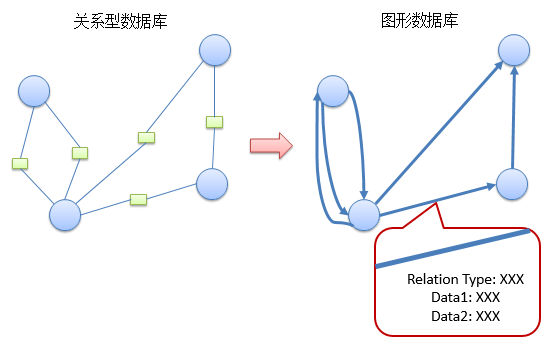


在表示关系的时候，关系型数据库和图形数据库就有很大的不同了：



从上图中可以看到，在需要表示多对多关系时，我们常常需要创建一个关联表来记录不同实体的多对多关系，而且这些关联表常常不用来记录信息。如果两个实体之间拥有多种关系，那么我们就需要在它们之间创建多个关联表。而在一个图形数据库中，我们只需要标明两者之间存在着不同的关系，例如用DirectBy关系指向电影的导演，或用ActBy关系来指定参与电影拍摄的各个演员。同时在ActBy关系中，我们更可以通过关系中的属性来表示其是否是该电影的主演。而且从上面所展示的关系的名称上可以看出，关系是有向的。如果希望在两个结点集间建立双向关系，我们就需要为每个方向定义一个关系。

也就是说，相对于关系数据库中的各种关联表，图形数据库中的关系可以通过关系能够包含属性这一功能来提供更为丰富的关系展现方式。因此相较于关系型数据库，图形数据库的用户在对事物进行抽象时将拥有一个额外的武器，那就是丰富的关系：



虽然在一个图数据库中常常拥有结点集的概念，但是它已经不再作为图数据库的最重要抽象方式了。甚至从某些图形数据库已经允许软件开发人员使用Schemaless结点这一点上来看，它们已经将结点集的概念弱化了。反过来，我们思考的角度就应该是结点个体，以及这些个体之间所存在的一系列关系。

但是并不代表可以随便定义各个结点所具有的数据。最为常用的一个准则就是：Schemaless这种灵活度能为你带来好处。例如相较于强类型语言，弱类型语言可以为软件开发人员带来更大的开发灵活度，但是其维护性和严谨性常常不如强类型语言。同样地，在使用Schemaless结点时也要兼顾灵活性和维护性。

这样我们就可以在结点中添加多种多样的关系，而不用像在关系型数据库中那样需要担心是否需要通过更改数据库的Schema来记录一些外键。这进而允许软件开发人员在各结点间添加多种多样的关系。

简单地说，单一事物应该被抽象为一个结点，而同一类型的结点被记录在同一个结点集中。结点集内各结点所包含的数据可能有一些不同，如一个人可能有不同的职责并由此通过不同的关系和其它结点关联。例如一个人既可能是演员，可能是导演，也可能是演员兼导演。在关系型数据库中，我们可能需要为演员和导演建立不同的表。而在图形数据库中，这三种类型的人都是人这个结点集内的数据，而不同的仅仅是它们通过不同的关系连接到不同的结点上。也就是说，在图形数据库中，结点集并不会像关系型数据库中的表一样粒度那么小。

一旦抽象出了各个结点集，我们就需要找出这些结点之间所可能拥有的关系。这些关系不仅仅是跨结点集的。有时候，这些关系是同一结点集内的结点之间的关系，甚至是同一结点指向自身的关系。

**三、neo4j数据和其他图数据库对比分析**

**1）TitanDB 数据库**

我们先研究了 TitanDB，它各项强大的功能和极佳的可扩展性一开始让我们非常振奋。可惜的是，TitanDB 的启动和维护都非常复杂，必须得从 Cassandra 或 HBase 后台运行。

我们关心的另一个功能是最终一致存储，它并不符合 ACID 原理。这表示，如果我们要长时间运行大型图形数据库，最后可能会出现不一致现象。

TitanDB 确实提供了一个基本可长期运行的流程，能够始终如一地穿行整个图形，以期探测和修复不一致问题。除了这些不一致之外，TitanDB 还可以作为不基于图形的本地存储之上的层。

**2）OrientDB 数据库**

接下来我们又了解了 OrientDB。OrientDB 启动起来似乎简单得多，还具备大量针对文档的功能。但从社区的评论来看，性能和可扩展性是个问题。另外，OrientDB 把自己宣传成多模式数据库 ——图形和 SQL。这种宣传缺乏对纯图形操作的针对性，让我很是忧心，我们不仅想要做图形，还要做好图形。

**3）发现 Neo4j**

然后我们发现了 Neo4j。Neo4j 可高度扩展，对节点、关系或索引的数量没有限制。同时 Neo4j 入门也相当简单，这对我们是很大的诱惑；在使用第三个数据库时，必须得迅速投入运行。

性能表现极佳，扩增也非常广泛，并且只专注于图形用例。Titan 确实提供映射（作为本地节点类型）支持，但我们知道，即使没有这一支持我们也可以继续下去。

总的来说，我们之所以选择 Neo4j，有以下原因：

* Highily scalable（高度可扩展）
* Easy to get started（易于上手）
* Good performance（良好的表现）
* Extensive documentation（丰富的文档）
* Adequate feature set（足够的功能集）

Neo4j 易于使用，设置和维护也很简单，实现甚至超出了我们的预期。它让我们超越了我们的概念点，非常非常迅速地投入运行和构建新事物。

##### Neo4j市场状况

1. **知识图谱的应用场景**

有时候我们在使用搜索引擎时， 我们的搜索词（Query）往往看起来更像是一个问题，比如“张三是从哪里毕业的”，这时我们需要搜索引擎直接给出我想要的结果，而不是一个网页排名(page rank) 。比如我在 google 中搜索“扎克伯格的妻子是谁”， 我需要是扎克伯格的妻子普莉希拉・陈的详细信息而不是一些包含了她信息的网页。

那么Google是怎么做到的呢？其实早在2012年Google 就已经在搜索中加入了知识图谱，用户可以通过Google 构建的知识图谱直接查询到结果，这种方式极大地提升了用户体验。然而对于 Google 来说处理起来也比较便捷，首先将“扎克伯格的妻子是谁” 这个 Query通过自然语言处理技术（NLP）处理成“扎克伯格”实体和”has\_wife"的关系，从已经构建好的知识图谱中查询， 然后将查询结果返回给用户。

然而就是这样的一个改动，从用户使用的角度来看已经从普通的搜索引擎变成了智能问答的系统，用户体验上升了一个层次。

我们可以将图领域划分成以下两部分：

* 用于联机事务图的持久化技术（通常直接实时地从应用程序中访问）。
* 这类技术被称为图数据库，它们和“通常的”关系型数据库世界中的联机事务处理（Online Transactional Processing，OLTP）数据库是一样的。
* 用于离线图分析的技术（通常都是按照一系列步骤执行）。这类技术被称为图计算引擎。它们可以和其他大数据分析技术看做一类，如数据挖掘和联机分析处理（Online Analytical Processing，OLAP）。

图数据库一般用于事务（OLTP）系统中。图数据库支持对图数据模型的增、删、改、查（CRUD）方法。相应地，它们也对事务性能进行了优化，在设计时通常需要考虑事务完整性和操作可用性。

目前图数据库的巨大用途得到了认可，它跟不同领域的很多问题都有关联。最常用的图论算法包括各种类型的最短路径计算、测地线（Geodesic Path）、集中度测量（如PageRank、特征向量集中度、亲密度、关系度、HITS等）。那么，什么样的应用场景可以很好地利用图数据库？

业内基于图数据库已经有了相对比较成熟的的解决方案，大致可以分为以下几类。

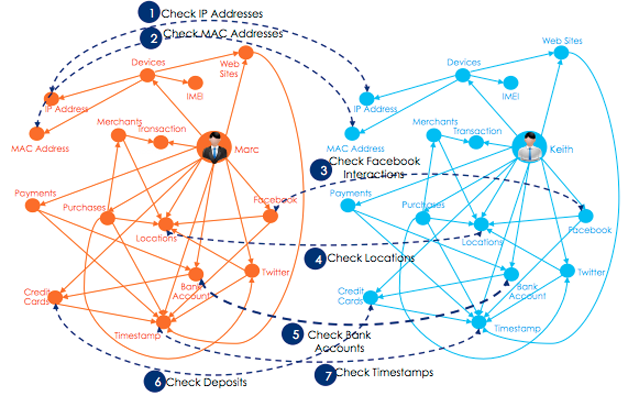
* 金融行业应用

反欺诈多维关联分析场景

通过图分析可以清楚地知道洗钱网络及相关嫌疑，例如对用户所使用的帐号、发生交易时的IP地址、MAC地址、手机IMEI号等进行关联分析。

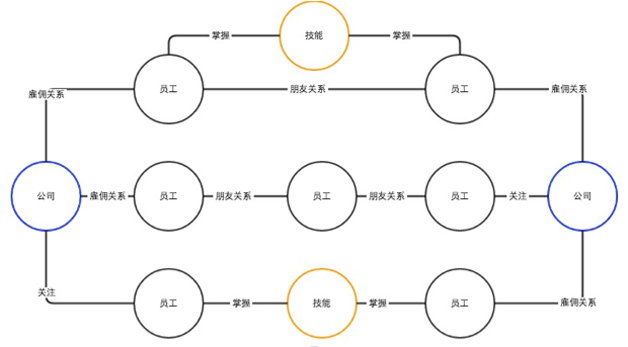
反欺诈多维关联分析场景

反欺诈已经是金融行业一个核心应用，通过图数据库可以对不同的个体、团体做关联分析，从人物在指定时间内的行为，例如去过地方的IP地址、曾经使用过的MAC地址（包括手机端、PC端、WIFI等）、社交网络的关联度分析，同一时间点是否曾经在同一地理位置附近出现过，银行账号之间是否有历史交易信息等。



* 社交网络图谱

在社交网络中，公司、员工、技能的信息，这些都是节点，它们之间的关系和朋友之间的关系都是边，在这里面图数据库可以做一些非常复杂的公司之间关系的查询。比如说公司到员工、员工到其他公司，从中找类似的公司、相似的公司，都可以在这个系统内完成。



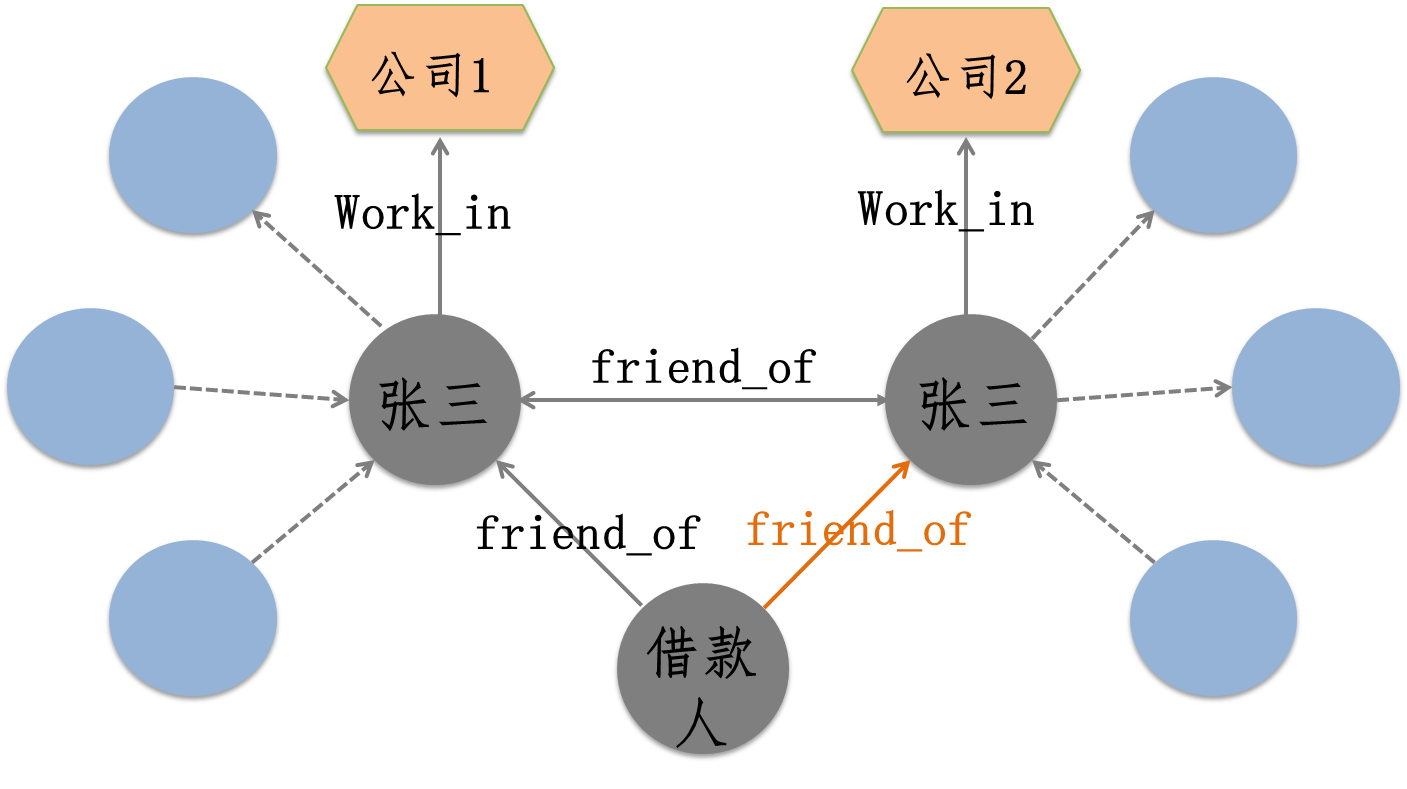
* 企业关系图谱

图数据库可以对各种企业进行信息图谱的建立，包括最基本的工商信息，包括何时注册、谁注册、注册资本、在何处办公、经营范围、高管架构。围绕企业的经营范围，继续细化去查询企业究竟有哪些产品或服务，例如通过企业名称查询到企业的自媒体，从而给予其更多关注和了解。另外也包括对企业的产品和服务的数据关联，查看该企业有没有令人信服的自主知识产权和相关资质来支撑业务的开展。

企业在日常经营中，与客户、合作伙伴、渠道方、投资者都会打交道，这也决定了企业对社会各个领域都广有涉猎，呈现面错综复杂，因此可以通过企业数据图谱来查询，层层挖掘信息。基于图数据的企业信息查询可以真正了解企业的方方面面，而不再是传统单一的工商信息查询。

马克斯·韦伯曾说“人是悬挂在自我编织的意义之网上的动物”，这句话从侧面说明人与人之间的关系是很复杂的，我们是否可以将复杂的人际关系进行一次挖掘呢？

首先人际关系实际形如一张网， 既然是网那么它一定具有一个特性，即网上两个相邻节点之间的路径损坏，并不一定影响整张网。比如 一张网（无向图）中相邻的 A 节点到 B 节点的路径“坏了”， 有极大可能找到另一条从 A 到 B 的路径，而不影响整张网。那么网的这个特性应该怎样应用到数据挖掘上来呢？我们来看一个知识图谱在银行风控系统中的一个例子。



我们可以根据借款人借款时填写的关系构建知识图谱， 如图借款人跟张三是朋友关系，跟李四是父子关系。当我们试图把借款人的信息添加到知识图谱里的时候，“一致性验证”引擎会触发。引擎首先会去读取张三和李四的关系，从而去验证这个“三角关系”是否正确。很显然，朋友的朋友不是父子关系，所以存在着明显的风险。这里的隐含关系挖掘可以借用通用的关系挖掘引擎，也可以自己实现隐含关系的挖掘引擎。通用关联关系的挖掘由于其通用性，通常难以保证对关系挖掘的正确性，通常是自己配置规则来确保关系挖掘的准确性。

知识图谱在银行风控中的作用还有很多，比如对失联借款人两度，甚至多度的关系挖掘来找到借款人等。 由此可见在“关系”越复杂的情况下，知识图谱越是能发挥它的作用。

**二、知识图谱的构建**

既然知识图谱这么有用， 那么怎样才能构建自己的知识图谱，怎样将传统的数据转化成知识图谱呢？

传统数据主要分成两种，格式化数据和非格式化数据。格式化数据转化成知识图谱时需要将格式化的数据映射成实体关系组，从而构建知识图谱。而非格式化的数据转化时比较复杂，通常采用算法抽取和程序抽取两种方式。

1. **算法抽取方式**

通过自然语言处理（NLP）技术对文本进行命名实体识别（NER），从非格式化的文本中识别出专有名词和有意义的短语并进行分类。比如上例中从“达观数据是一家人工智能公司”这段文本中识别出”达观数据”和“人工智能公司”这两个实体以及“是”这个从属关系，这样我们就可以通过”达观数据” “是” “人工智能公司” 这个实体组来构建知识图谱。由于目前NER识别技术还不够成熟，通常我们会对 NER 识别的实体进行人工矫正，确保所识别实体的准确性。

1. **程序抽取方式：**

在处理实体识别非格式化数据的过程中我们经常会碰到半格式化的数据，比如一段简历的文本，文本中经常会包含，姓名：XXX，公司名：XXX 等格式，遇到这样的半格式化文本，我们也可以采用正则等方式来抽取，确保知识图谱构建的完整性和准确性。