neo4j

全球领先的图数据库

技术综述

写给关系型数据库开发人员的

图数据库权威指南

目　录

其实并不像其名称听上去的那样，关系型数据库并不擅长处理今天高度关联的数据。其根本原因是关系型数据库并不具备足够强大的存储和管理数据项之间关系的能力。

写给关系型数据库开发人员的

图数据库权威指南

第一章、

为什么关系型数据库不是万能的?

关系型数据库(RDBMS)是非常强大的技术工具。

自从上世纪八十年代起，关系型数据库就是大多数软件和应用系统的发电站，而且这一状况今天仍然在继续。最初设计时，关系型数据库针对的是规范的、类似纸张表格的表状数据结构，而且它的确非常出色地满足了这种需求。对于适合的应用案例和架构，关系型数据库是存储和组织数据的最佳工具。

关系型数据库在表中存储高度结构化的数据。表通常具有事先定义好的列和包含相同类型信息的行，它们要求开发人员和应用对使用到的数据都必须严格地进行定义。

然而，在大数据时代，数据的格式和类型更加多样、数据量更大、变化更加频繁 、更新更加快速，而更重要的是数据之间的关系和联结越来越得到重视。今天对数据处理的需求和应用已经不再满足于简单的表状结构。

关系型数据库并不擅长存储关系

其实并不像其名称听上去的那样，关系型数据库并不非常擅长处理今天高度关联的数据。其根本原因是关系型数据库并不具备足够强大的存储和管理数据项之间**关系**的能力。

如果我们回顾一下历史，关系型数据库的名称来自于E.F. Codd的关系代数中“关系”的数学概念，它具有非常特定的含义。这个名称其实和存在于现实世界中事物之间的关系没什么太大联系。

一直以来，开发人员在做的都是怎样将数据存储到关系模型的列和行中。 而这并不是现实生活中数据存在的实际形式。实事求是地说，数据更应被看作是对象， 以及存在于无数对象间的连接。

这种复杂的、存在于现实生活的数据在规模、产生速度和多样性上飞速增长 。与此同时，数据之间的关联也在以更加快速的节奏增长，这些联结往往包含着比数据本身更有价值的知识。

这就是新问题产生的根本原因：关系型数据库并非为保存和处理这种丰富的、存在于数据间的联系而设计。

今天的企业和机构由于没有适用于处理高度关联的数据的技术手段，而错失低风险、高利润、数据驱动的决策机会。

其带来的主要问题就是，今天的企业和机构由于没有适用于处理高度关联的数据的技术手段，而错失低风险、高利润、数据驱动的决策机会。

现今的软件应用都需要适应敏捷开发的要求

今天，所有的开发团队都不得不面对变化频繁的业务和用户需求，而这些变化往往要求对已有的数据架构和视图进行修改。

数据库管理员 (DBAs)和开发人员时常要应对来自业务部门改变和增加数据库元素及属性的要求，以满足经常变化的业务，例如存储最新社交平台的数据。 然而这种经常性的数据库模式的改变对关系型数据库是存在问题的，而且改变的代价昂贵。

其主要原因是关系型数据库面对变化并不能很好地适应。固定的数据库模式只适用于能够事先明确的业务问题，在不需要经常改动的情况下才能运行稳定。

缓慢和昂贵的数据库模式重设计同样对敏捷软件开发带来负面影响，它使得团队无法进行快速的创新，错失巨大的市场机会。

结论就是，关系型数据库不是为适应灵活快速的业务发展而设计的。

对数据关联的查询是怎样拖垮关系型数据库性能的？

尽管有了更加先进的计算技术、更快的处理器和高速的网络，某些类型的关系型数据库应用却越来越慢。这种性能的下滑有一些众所周知的症状(参见下文关于“SQL查询的限制”部分)，但究其根源总无外乎一个共同因素：对数据关系的查询。

虽然关系型数据库并非为处理连接的数据而开发和优化，但是任何试图从数据间关联中寻找答案的查询，例如推荐引擎、欺诈检测或者社交网络图，都必须使用大量的数据库表间的连接(JOIN)操作。

在关系型数据库中，对其他表中记录的引用是通过定义指向另一表中的主键属性的外键字段来表示的。参见下面图1中的一个例子。

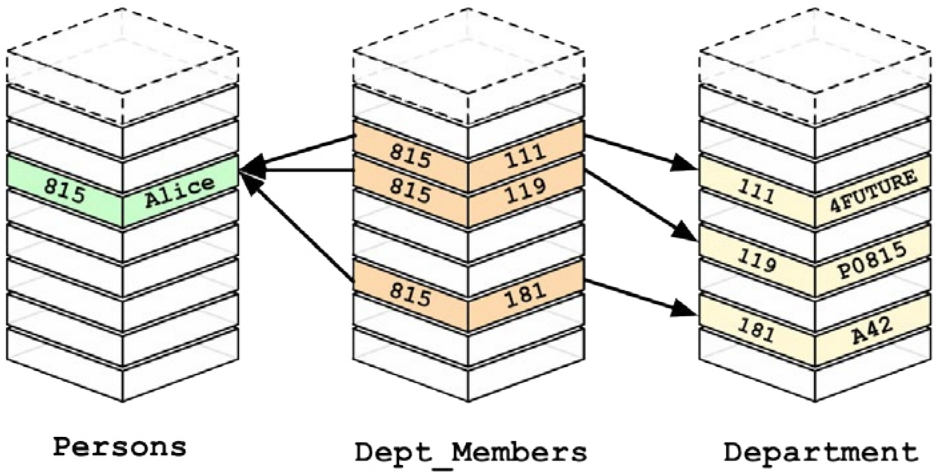

图1: 在一个关系型数据库中使用JOIN连接**Persons**和**Departments**表，这里使用了外键约束。

表间的引用由限制来得到强制执行，但是这仅仅在引用不是可选的情况下。在执行查询时，JOIN操作比较主表和被连接表中每条记录的主键和外键的值。多数情况下，键字段建有索引以提高这种的比较操作的性能，但是即便如此，这样的操作还是会消耗大量的计算能力和内存，并且随着查询复杂度的增长而呈指数级增长。

于是，在关系型数据库中对联结的数据进行建模和存储几乎不可避免地带来极端的复杂性。这种复杂性包括长达成百上千行的SQL查询语句，有时仅仅实现了简单的功能。查询性能因着查询的复杂性、数据间关系的数量和层次，以及数据库的规模的增长而越来越糟糕。

今天的软件应用都需要满足实时处理、始终可用的要求。传统的关系型数据库在要求处理大量数据间关联的应用中已经不能胜任。

关系型数据库应用受限于SQL的五大征兆

多数关系型数据库应用在其能力范围之内都可以运行地很好。但是，有些则因着数据库本身的限制而运行缓慢，特别是当关系型数据库被用来处理高度联结的数据时。

以下是试图用关系型数据库解决数据联结型问题时会遇到的五大典型征兆：

1. 有大量的连接JOIN操作

当查询中包含了过多的表连接的时候，复杂性和计算资源消呈爆炸性地增长。这相应地增加了查询的执行时间。

2. 有大量的自连接Self-JOIN操作(也称作递归连接)

自连接查询在层次和树状的数据库结构中非常常见，然而，重复连接表自身以达到遍历关系的目的是非常低效的。事实是，我们见过的这世界上最长的SQL查询就包含递归的自连接。

3. 频繁变更的数据库模式

现今的时代，业务的灵活可变性是在激烈竞争中制胜的关键。而实际情况是，业务部门要求做出变化的意愿经常被DBA们泼上冷水，因为关系型数据库并非为适应频繁变化的数据模式和视图而设计的。对数据库模式变化的要求变得平常，说明数据和需求在快速的变化，这就要求一个更加灵活可变的数据模式。

4. 运行缓慢的查询(尽管进行了大量的性能调试)

数据库管理员们尝试使用教科书上提到的所有秘诀来加速查询的运行，但是许多SQL的执行速度仍然无法达到期待、以满足应用的需求。另外，为提高查询性能而采用的“去范式化”，或称“扁平化”(denormalizing)的数据模型会对数据质量和更新方式带来负面的影响。

5. 预先计算查询结果

因为查询的速度太慢，许多应用不得不用历史数据事先计算好结果，这实际上是用昨天的数据来获得原本需要用实时数据才能获得的查询结果。更糟糕的是，系统往往不得不计算100%的数据，即便只有1-2%的数据是真正需要 的。

关系数据库的替代方案

前面提到过，关系型数据库有适合它的应用类型。对于结构规范、事先可以定义的模式，关系型数据库是完美的技术工具。

但是就像我们已经看到的，关系数据库并不总是最好的。那些需要从联结的数据中挖掘出内在洞察的应用就无法依靠关系数据库来高效解决。

今天的大数据呈现的规模、速度和多样性，以及数据间的关联，需要一种从**根本上能够存储和管理联结数据的解决方案**。而图数据库正是为这一目的而设计的。下面就让我们来看看图数据库。

第二章、

什么是图数据库?

我们已经知道关系数据库不再胜任大数据时代的在规模、速度和多样性等方面的变化，那么什么才是适用的替代方案呢？

目前已经有其他的数据库选项，包括一系列的 NoSQL数据存储(不仅仅是SQL，Not Only SQL)，但其中没有哪个是专门为处理和存储数据间关联而设计的，除了一种， 那就是图数据库。

图数据库能给现有技术架构带来的最大价值，是图数据库对数据之间的关系提供与单个数据同样的存储支持(First-Class Citizen)。

例如，图数据库的早期采纳者能够在数据之间的关联之上重新定义业务架构。这些公司今天已经成为业界的领袖，诸如：LinkedIn , Google, Facebook 和PayPal。

作为图数据库的先行者们，这些公司都从无到有创建了他们自己的图数据处理技术。幸运的是，对于今天的开发人员，从头创建图数据库已经不再必要，已有一个现成的图数据库产品可以采用，那就是Neo4j。

让我们再看看为什么应该采用图数据库来构建下一代基于联结数据的应用。先熟悉一下图和图数据库的基本定义。

如果曾经使用过关系数据库，那么理解图数据库只是小菜一碟。

图是什么?

图数据库基于图计算理论。不过，为了理解图数据库，我们不必非得理解神秘的、数学理论中的图。相反，如果曾经使用过关系数据库，理解图数据库只是小菜一碟。

第一件要知道的事：图(Graph)，在数学理论中，和图表(Chart)是不一样的，所以不要把图想象成一个条形图或线形图。

图应当被想象成一个由节点和边连接起来的网络，例如一个思维导图(Mind Map)，像右图所示。

图2、一个简单的欺诈团伙图，其中的节点共享联系信息。

图有两个基本元素：节点和连接节点的关系。

每一个节点代表一个实体，例如某人、某地、某事；每一个关系代表两个节点是如何相互关联的。举一个例子，有两个节点“蛋糕”和“甜食”，它们之间的关系可以是“是一种。。。”（is a type of），方向是从蛋糕到甜食。

这种通用性的结构允许对所有的情况进行建模，从一个道路系统，到连接到网络的设备，或者一群人的医疗记录，以及任何可以用关系连接的事物。

什么是图数据库?

图数据库是一个在线数据库管理系统，用来执行创建、读取、更新和删除（CRUD）图数据模型中的数据。图数据库通常也是交易型系统（OLTP）。因此，图数据库针对交易处理性能进行了优化，而且优先确保交易的完整性和系统的可用性。

与其他数据库不同的是，图数据库最优先处理数据间的关系。这意味着应用无须依靠外键来推导数据间的关联，或是依靠外部的系统来处理数据，例如MapReduce。

与其他数据库所不同的是，图数据库最优先处理数据间的关系。这意味着应用无须依靠外键来推导数据间的关联，或是依靠外部的系统来处理数据，例如MapReduce。

通过将节点和关系简单抽象成连接的结构，图数据库使得建立与现实问题领域更加接近的复杂的模型成为可能。

对图数据库技术而言，有两个重要的特性：

本地（ative）图存储

有些图数据库使用本地图存储，就是针对图的特性而专门设计的存储模式，而其它数据库则使用关系型或面向对象型数据库来存储图数据。非本地的图存储有很大的潜在性能瓶颈，尤其是在数据规模和查询复杂度显著增加的时候。

图处理引擎

本地图处理，我们称之为“无需索引的邻接关系”（index -free adjacency），是处理图数据最有效率的手段，因为数据的连接在图数据库中物理地被保存下来。其他非图的数据库以其他方法处理，无法针对图数据结构而优化CRUD操作。

使用图数据库有哪些优势?

图数据库是专门为处理高度联结的数据而设计建造的。当今是大数据的时代，数据拥具有极大的规模和高度的内在联系，而图数据库为实现“可持续的竞争优势”（ustainablecompetitiveadvantage）提供了巨大的机会。

在将图应用到现实世界中的问题上，并兼顾实际的技术和业务限制，许多企业出于一种或多种不同原因最终选择了图数据库：

从分钟到毫秒的性能提升

查询性能和响应速度是许多组织在建立其数据平台时首要关注的问题。在线交易系统，尤其是大型Web应用程序，必须在毫秒内对用户请求作出响应，才能确保客户不会因为失去耐心而离开。在关系型世界中，随着应用程序数据集大小的增长，复杂连接(JOIN)的局限性开始显现出来，性能迅速下降。使用无索引邻接关系，图数据库将复杂的JOIN转换为快速的图遍历，因此无论数据集的总体大小如何都可以保持毫秒级的性能。

显著加快开发周期

图数据模型解决了几十年来困扰软件开发领域的阻抗失配问题(Impedance ismatch)，从而减少了在对象模型和表格关系模型之间来回转换的开发成本。更重要的是，图模型同样消除了技术和业务领域之间的阻抗失配。领域专家 、架构师和开发人员可以使用共享的图模型来讨论和描绘核心的业务领域，然后将其纳入应用程序本身。

快速响应业务变化

成功的应用很少保持不变。商业状况、用户行为以及技术和运营基础架构的变化推动了新的需求。 过去，这要求组织进行细致和长时间的数据迁移，包括修改模式、转换数据和维护冗余数据以同时支持新的和已有的功能。

使用图数据库开发完全符合当今灵活的、测试驱动的开发实践，允许图数据库与其他应用程序和任何不断变化的业务需求同步发展。数据团队可以在不危及已有功能的情况下，将新的图结构添加到已有的图数据库中，而不是试图在一开始就对数据库进行详尽和完美的建模(那其实是不可能做到的)。

企业级数据库平台

在关键业务应用中，数据库必须具有强大的可扩展性，并且通常需要支持事务性。尽管图数据库相对新颖、应用不如RDBMS成熟，但也有图数据库提供了当今大型企业所需的所有功能(例如Neo4j)：

- ACID事务性
- 高可用性
- 读取操作的水平可扩展性
- 存储数十亿个实体

这些特性一直是促使大型企业采用图数据库的重要因素。图数据库不仅仅是中等规模的、离线的、或部门级应用，它是能真正改变整个企业的业务模式。

图数据库有哪些常用的应用场景？

虽然图数据库首先应用在消费者网络(Facebook, LinkedIn, Twitter)的社交应用中，但它们的应用场景远远超出了社交空间。当今的企业组织以多种方式使用图数据库技术，其中包括以下六种最常见的用例：

- 欺诈检测
- 实时推荐引擎
- 主数据管理(aste ata ana ement)
- 网络和IT营运
- 身份和访问管理
- 基于知识图谱的智能搜索

有关图数据库技术用例的更多信息请参阅技术白皮书《图数据库的五大经典应用案例》。

第三章、
数据建模的比较: 关系型数据库和图数据库

在某些方面，图数据库就像新一代的关系数据库，但图数据库提供对“关系”的最高级别支持，传统关系型数据库则通过外键表示隐含的关系连接。

图数据库模型中的每个节点（来自实体或属性）直接在物理存储上包含一系列关系记录，这些记录表示该节点与其他节点的关系。 这些关系记录按类型和方向进行组织， 并可以包含其他属性。

这种将关系预先**物理存储**在数据库中的能力使得像Neo4j这样的图数据库能将查询性能由原先的数分钟提高到数毫秒。

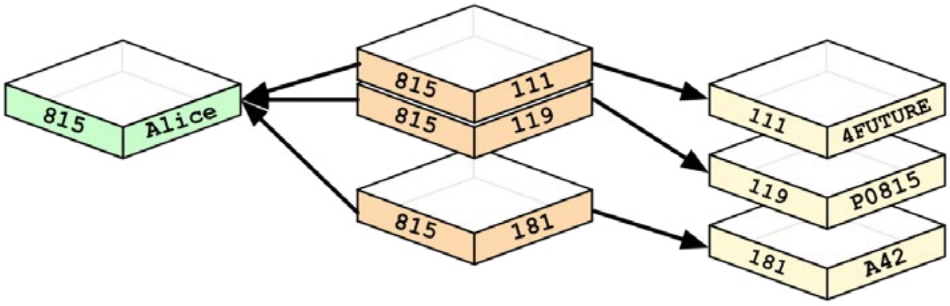

图3：以图的方式表示在关系数据库中，Person和Department表之间用外键表示的关系。

在图数据库中，无论何时运行类似JOIN的操作，数据库都会使用此列表并**直接**访问连接的节点，而无需进行昂贵的搜索和匹配计算。

这种将关系预先**物理存储**在数据库中的能力使得像Neo4j这样的图数据库能将查询性能由原先的数分钟提高到数毫秒，特别是对于JOIN频繁的查询，这种优势更加明显。由此产生的数据模型比使用传统关系数据库或其他NoSQL数据库生成的数据模型要简单得多，而表现力更强。

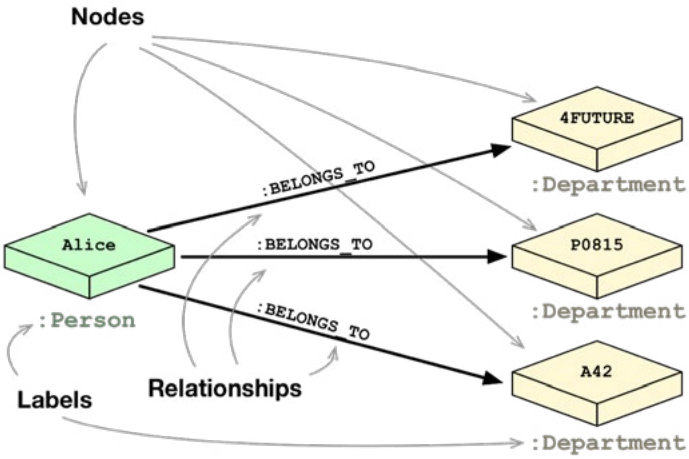

图4：我们原来的Persons表和Departments表中数据的图数据模型。节点和关系已经取代了我们的表、外键和JOIN表。

图数据库支持非常灵活和精细的数据建模，允许以简单直观的方式建模和管理含义丰富的领域模型。

图数据模型按照现实世界的形式保留数据：小规模、规范化、有丰富关联的实体。这使得可以从任何可以想得到的兴趣点查询和查看数据，并且支持许多不同的用例(有关更多信息，请参阅第2章)。

细粒度的模型意味着集合周围可以没有固定边界，所以更新操作的范围由应用程序在读取或写入操作时指定。事务特性保证对一组节点和关系的更新被包含在一个原子化、一致性、彼此隔离并且持久化的(ACID)操作中完成。

像Neo4j这样的图数据库完全支持关系数据库中的事务概念，包括异常终止后的预写日志和恢复，所以对已提交给数据库的数据，永远不会发生丢失的情况。

如果在关系数据库建模方面曾经有丰富的经验，想象一下如果有一个一致的、规范化的实体关系图，那将是多么简洁和美观。那将是一种简单易懂的模型，可以快速地与同事和领域专家在白板上进行交流。图模型就是这样，它是关于领域知识的清晰模型，让所有项目参与人员能够专注在想要有效支持的用例上面。

下面，我们以一个组织机构的领域模型为例，来展示它在关系数据库和图数据库中会怎样建模。先来看看关系数据库模型：

图数据库支持非常灵活和精细的数据建模，允许以简单直观的方式建模和管理含义丰富的领域模型。

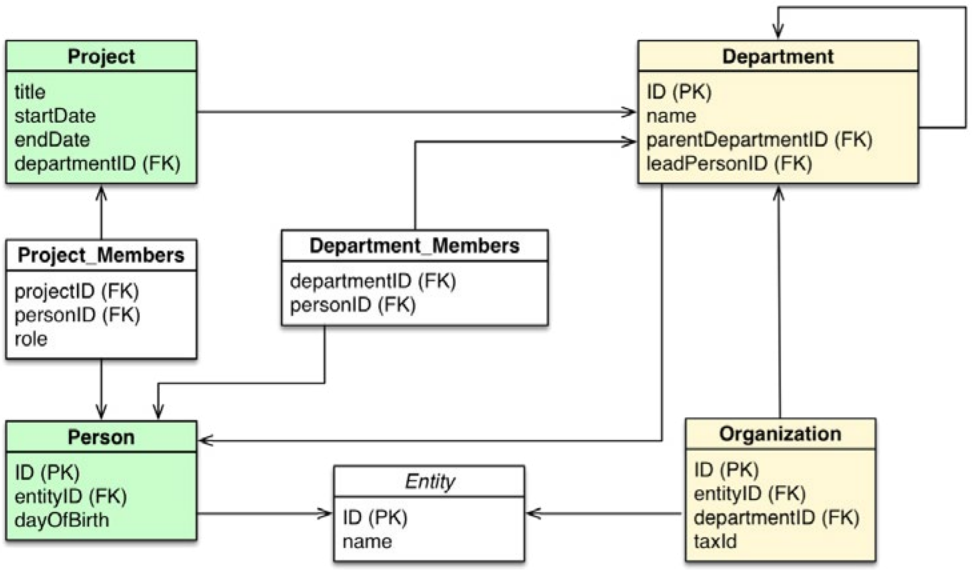

图5：在有多个部门(e aDt e t)的组织中(Da i atio)，关于人员(e o)和项目(oe t)的领域数据模型实例。

如果将上面这个关系数据库的模型转换为图数据库模型，可以通过以下步骤来进行：

- 每个实体表（a □e）由节点（ e）上的标签（□a e□）表示；
- 实体表中的每一行（ e □）都是一个节点；
- 这些表上的列成为节点的属性（ □ e□t□）；
- 删除技术主键，但保留业务主键；
- 为业务主键添加唯一性约束，并为需要频繁查找的属性添加索引；
- 将外键转换成节点间的关系（□e□atio i），然后删除它们；
- 删除存储默认值的属性，因为不需要存储这些数据；
- 扁平化（ enormalized）和重复的表数据需要提取出来成为单独的节点，以得到更清晰的模型；
- 一些列需要被定义成数组类型的属性，例如email1，email2，email3；
- 连接关系表（通常用来描述多对多关系）中的每条记录转换为关系，并且这些表上的列成为关系的属性。

在完成了这些步骤来简化关系数据库模型后，我们就得到以下的图数据模型：

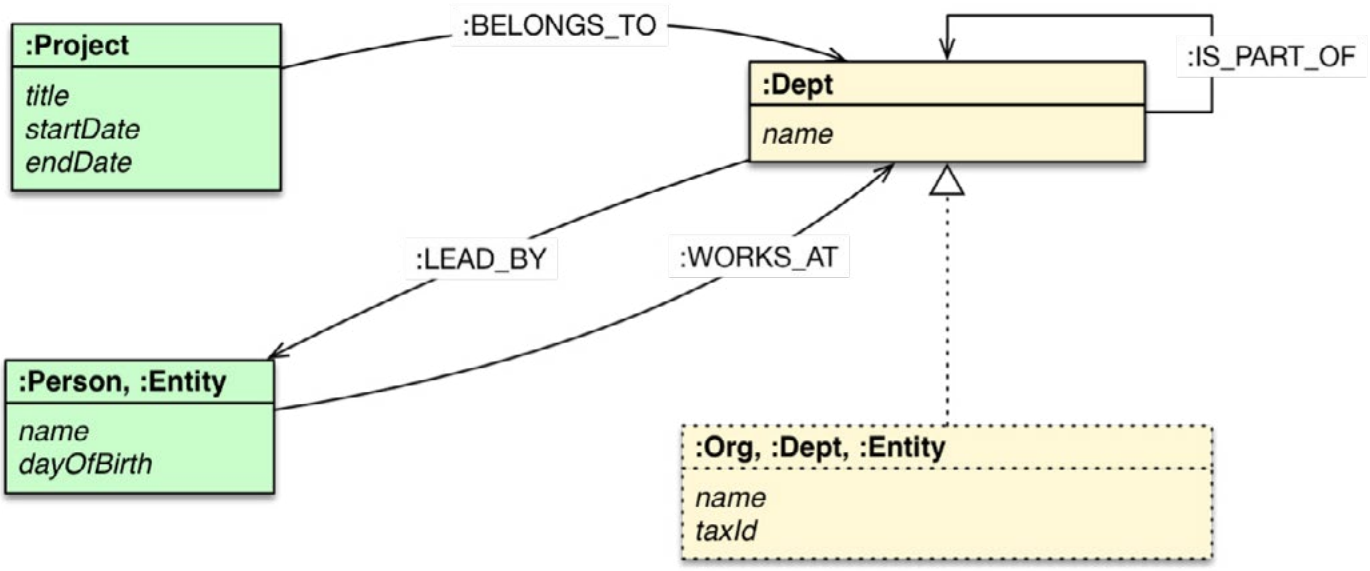

图6：在有多个部门（Department）的组织中（Organization），关于人员（Person）和项目（Project）的领域图数据模型。在图模型中，所有关系数据库中的**连接表**现在都变成了节点间的**关系**。

上面的这个例子只是关系型模型和图数据模型的一个简单比较。接下来让我们深入探讨一个来自真实世界的、应用更广泛的例子。

关系型与图数据建模案例研究：数据中心管理领域模型

为了展示图数据建模的真正威力，我们将看看如何使用关系模型和图模型的基本方法对某个领域进行建模。你可能已经很熟悉RDBMS数据建模技术，因此下面的比较将突出一些相似之处，以及更多的差异。

特别地，我们将展示从概念图模型转换到物理图模型是多么容易。与关系型数据模型相比，图模型是如何极大地消除了化我们用来表示的数据模型与现实世界之间的差异。

为了便于比较，我们将研究一个简化了的数据中心管理领域模型。在这个领域模型中，多个数据中心支持许多属于不同用户的应用，应用使用基础架构的不同部分，包括虚拟机到物理的负载均衡设备。

图为描述领域知识提供了更清晰的模型，它专注于要描述和解决的实际问题。

下图是一个小型数据中心的领域模型：

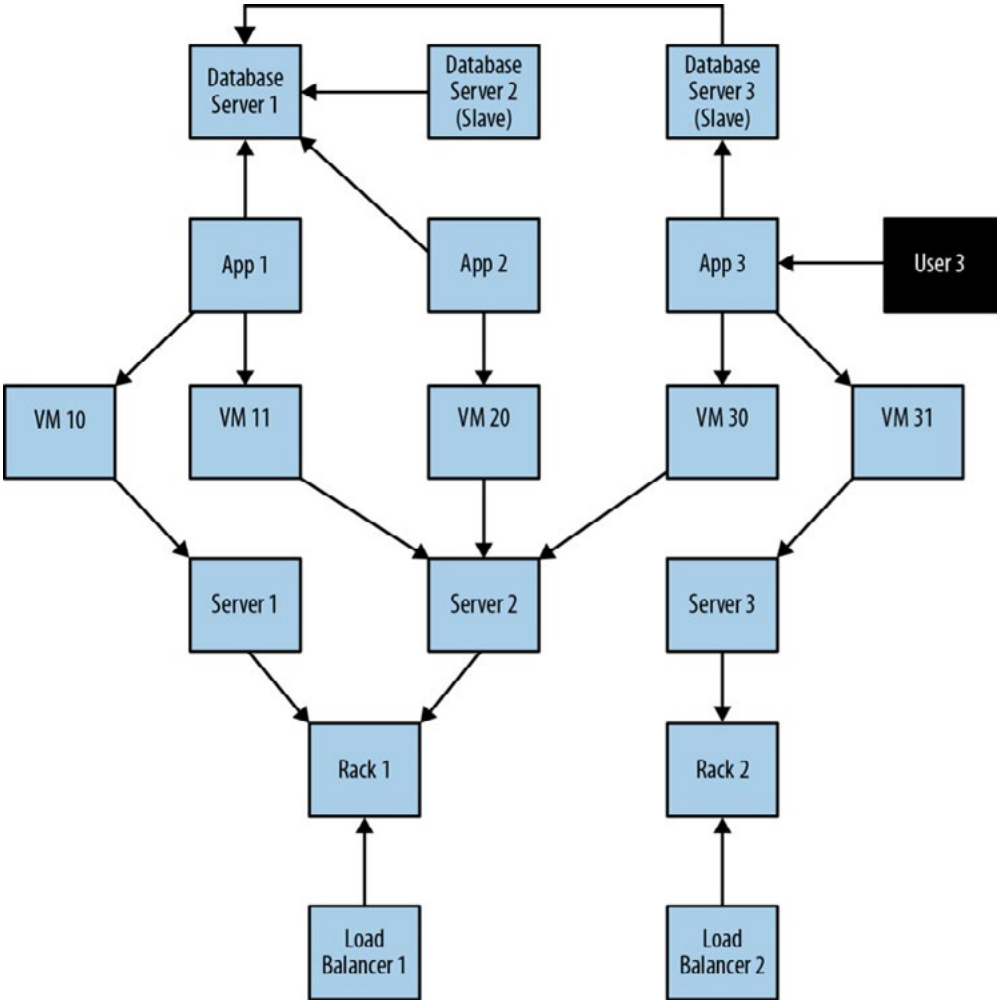

图7 一个部署了多个应用程序的小型数据中心领域模型

在上面的这个例子中，我们看到几个应用程序和支持它们所需的数据中心基础结构的简化视图。由节点App 1、App 2和App 3表示的应用程序依赖于标记为数据库服务器的Database Server 1 、Database Server 2和Database Server 3的数据库集群。

虽然用户在逻辑上依赖于应用程序及其数据的可用性，但还有其他物理用户和应用程序之间的基础设施：包括虚拟机(VM 10,VM 11, VM 20, VM 30, VM 31)，物理服务器(Server 1, Server 2, Server 3)，服务器机架（Rack 1, Rack 2）和负载均衡设备（Load Balancer 1, Load Balancer 2），它们是应用程序直接面对的访问入口。

当然，每个组件之间都有很多网络元素：电缆、交换机、配线架、NIC(网络接口控制器)、电源、空调等等，所有这些都可能在任何时候出现故障。为了说明完整的应用场景，我们假设应用程序App 3有一个访问用户User 3。

作为数据中心的运营商，有两个主要关注点：

• 持续提供满足(甚至超过)服务级别协议(QoS)的性能，包括执行前瞻性分析以确定单点故障的能力，以及执行事后分析以迅速确定任何顾客对服务可用性投诉的原因；

• 对消耗的资源计费，包括硬件成本、虚拟机、网络通信流量，甚至软件开发和运营成本(因为这些都是我们在这里看到的系统的逻辑扩展)。

如果我们正在构建数据中心管理解决方案，那么需要确保底层数据模型让我们能以最有效解决主要问题的方式存储和查询数据。我们还希望能够随着应用程序的变化，或是数据中心的物理布局发生演变、虚拟机实例迁移后，能够方便地更新底层模型。

基于这些需求和约束，让我们对比一下关系模型和图模型。

创建关系型数据模型

关系数据建模的第一步与任何其他数据建模方法基本相同：理解并就领域中的实体达成一致，包括它们之间的相互关系、以及决定其状态转换的规则。

关系数据建模的第一步与任何其他数据建模方法基本相同：理解并就领域中的实体达成一致。

这种初始阶段通常是非正式的，领域主题专家和数据架构师之间在白板上产生大量草图和讨论。这些谈话通常会产生如上图7所示的草图(它恰好也是一张图)。

下一步是将此初始白板草图转换为更严格的实体关系图(E-R图)，这又是另一个图。在将概念模型转换为逻辑模型的过程中会使用更严格的符号，这也为我们提供了第二次机会来优化领域的词汇表，以便与关系数据库专家共享。

(值得提到的是熟练的RDBMS开发人员通常不用中间的E-R图就直接跳到表设计和范式化过程中)。

下面是我们的示例E-R图：

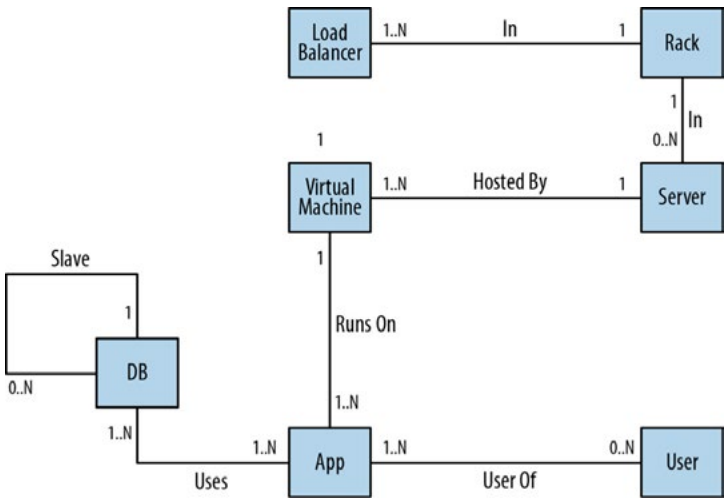

图8：数据中心领域模型的实体关系(E-R)图。

关于E-R关系图的说明

尽管也是图，E-R图立即凸显了关系模型的缺点：E-R图只允许实体之间的单一无向关系。在这方面，关系模型无法胜任实体之间关系众多、且语义丰富的现实世界领域问题的建模。

完成一个逻辑模型后，现在是将它映射到表和关系的时候了。表和关系通过正则化以消除数据冗余。 在很多情况下，这一步像将E-R图转换成表格形式一样简单，接着通过SQL命令将这些表加载到数据库中。

但即使是最简单的情况也能反映出关系模型的一些特质。 例如，在下图中我们看到，大量的预料之外的复杂性以外键约束（所有标注为[FK]的属性）的形式被加入模型。这些外键用来支持一对多关系。还有连接JOIN 表（例如AppDatabase），用来支持多对多关系 。所有这些，都发生在我们还未添加一行真正的用户数据之前。

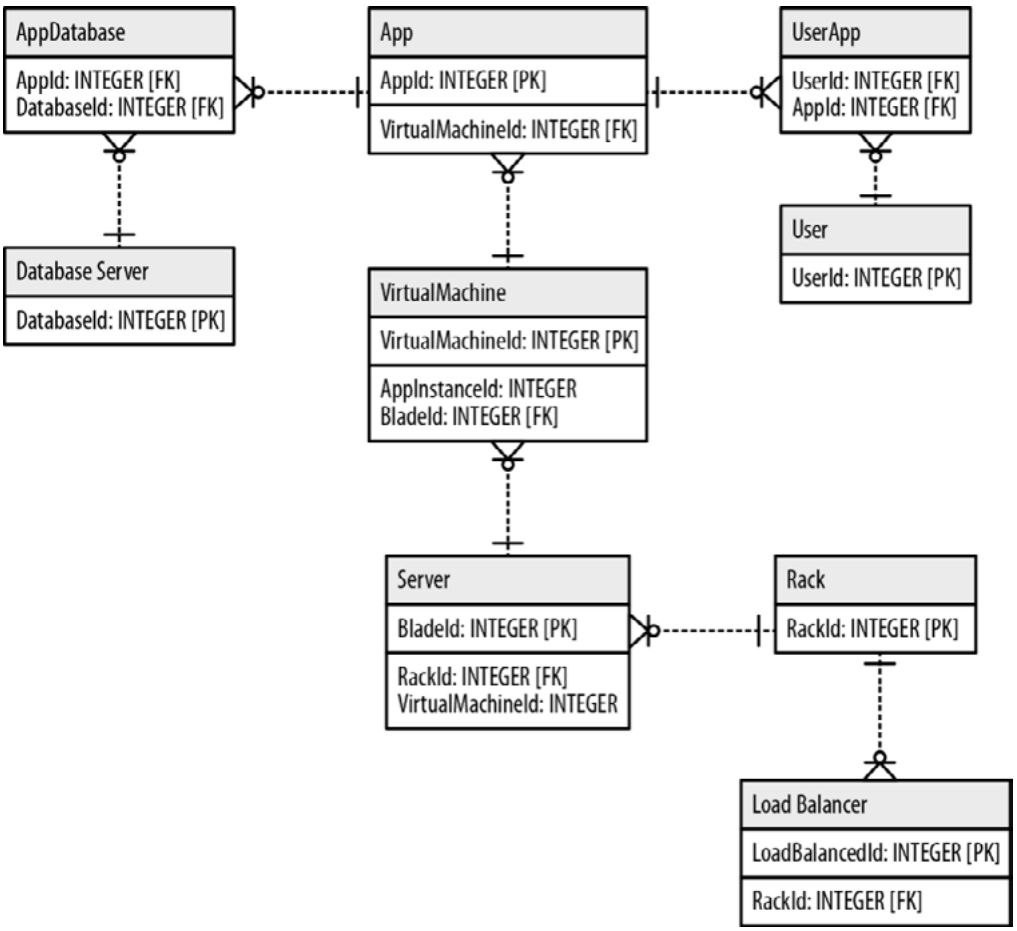

图9 关于数据中心的完整关系数据模型

这些约束和复杂性都是模型级的元数据，它们存在的唯一目的是为了在查询时指定表之间的关系。然而，这种结构化数据的存在也带来了局限性：因为它服务于数据库、而不是数据的使用者，因此往往最终变成令人迷惑和含义模糊的数据模型。

关系数据模型扁平化带来的问题

到目前为止，我们建立了一个规范化的关系数据模型，它相对忠实地反映了领域数据，但我们的设计工作尚未完成。

关系范式的挑战之一是规范化的模型通常不足以满足实际应用中对查询速度的要求。理论上，规范化模式可以回答我们对应用领域提出的任何类型的查询，但实际上模型必须进一步修改以适应特定的访问模式。

换句话说，为了使关系数据库的性能满足应用程序的需求，我们必须放弃一些的领域数据的关联性，并接受这样的事实：必须改变用户的数据模型，以适应数据库引擎、而不是用户的需要。这种方法称为去规范化，或称扁平化（ enormalization）。

去规范化涉及复制数据（在某些情况还是相当规模的），以获得更高查询性能。

例如，考虑一批用户及其联系人详细信息。一个典型的用户通常有几个电子邮件地址，通常会将其存储在单独的EMAIL表中。但是，为了减少连接(JOIN)两个表带来的性能开销，通常会在USER表中添加一列或多列以存储用户最重要的电子邮件地址。

要让项目中的每个开发人员都能理解去规范化的数据模型，以及它如何映射到以领域为中心的代码，并不都是一项轻松的任务。

更多的情况是，开发团队转向RDBMS专家，将规范化模型转换成扁平的数据模型，确保RDBMS层和物理存储层的特性保持一致。而为了实现这些，会造成大量的数据冗余。

在关系数据模型中实现快速变更的代价

或许我们会觉得规范化/扁平化的过程是可以接受的，毕竟它只是一次性的任务。不管怎样，这项前期工作的成本在系统的整个生命周期中都会得到回报，对吗？错误。

虽然这种一次性、事先考虑的想法很有吸引力，但它不符合当今敏捷开发流程的实际情况。如今的系统变化频繁，不仅在开发过程中、而且在其在线运行期间。

尽管大多数系统大部分时间都在生产环境中运行，这些环境却很少是一成不变的。商业需求会变化，监管要求在演变，所以我们的数据模型也必须及时跟进。

调整关系型数据库模型需要一个称为迁移的结构变化。迁移为数据库重构提供了一种结构化、分步骤实现的方法，因此它可以发展以满足不断变化的需求。然而，与代码重构（通常需要几分钟或几秒钟）不同，数据库重构可能需要几周或几个月才能完成，而数据模式更改还会造成应用下线。

去规范化的关系模型的底层问题是，它事实上与现今业务应用的迅速发展相抵制。正如我们在这个数据中心例子中看到的那样，从头到尾对初始的白板模型施加的变化，造成了概念世界和数据存储方式之间的不可逾越的鸿沟。

这种概念和关系模型之间的不协调性阻碍了来自业务部门和其他非技术参与者在系统演化方面的进一步合作。因此，应用程序的发展永远落后于业务发展。

现在我们已经仔细研究了关系数据建模过程，下面再看看图数据建模方法。

创建图数据模型

正如上面所看到的，关系数据建模事实上将应用程序的存储模型与领域专家描述的概念脱离开来。

关系数据库，由于其严格的模式和复杂的建模特征，并不是支持快速变化的最佳工具。我们需要的是一个与领域紧密结合的模型，不以牺牲性能为代价；支持演化，并且在数据快速变化和增长时依旧保持数据的一致性。

这样的模型就是图模型。那么，图的数据建模过程是如何不同的呢？

在图数据建模的开始阶段，工作与关系方法类似：使用白板画出高度概括的模型草图，描述并就初始的领域模型达成一致。然而在那之后，数据建模的方法就非常不同了。

再来看看我们的案例。下图是在白板上描绘的数据中心领域模型：

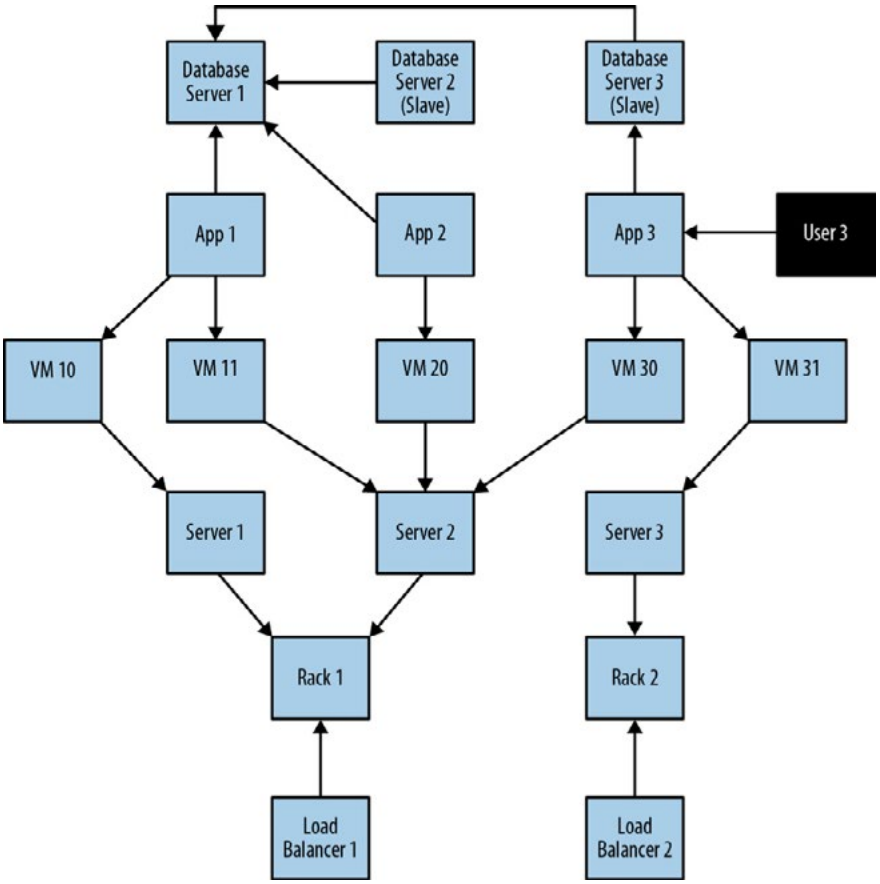

图10：样例数据中心领域模型，包含多个部署的应用程序。

在图数据建模中，下一步就是简化我们已有的图结构，而不是将模型转换为表格。这一步骤旨在让数据模型更准确地表达应用的目标。具体来说，我们将相关角色定义为为标签(□a e□)，实体的属性(Att□ te)定义为标签的属性(□ e□t□)，与相邻实体之间的关系定义成关系(e□ati i)。

在第一轮的领域建模中，我们通过增加属性和关系来丰富模型，生成了一个适合数据需求的图模型。也就是说，我们构建了一个模型让应用程序能够从其数据中询问各种问题。

为了进一步丰富正在开发中的图数据模型，我们需要确保正确的上下文语义。这通过在节点之间创建命名的和定向的关系，就可以捕捉领域知识的复杂结构。

从逻辑上讲，这就是全部所需要做的。没有表格，没有规范化，没有去规范化 。一旦我们准确地表达了领域模型，将其在图数据库中实现反而是最轻松的事情。

图数据建模的下一步就是简化我们已有的图结构，而不是将模型转换为表格。没有表格，无须范式化或是去范式化。一旦我们准确地表达了领域模型，将其在数据库中实现反而是最轻松的事情。

使用图数据库，在白板上绘制的内容就是在数据库中存储的内容。

没有陷阱！

有什么陷阱吗？

听上去似乎太完美，那么，有什么陷阱吗？没有！使用图数据库，在白板上绘制的内容**就是**在数据库中存储的内容。

就这么简单。

在添加属性、标签和关系后，数据中心领域的结果图模型如下所示：

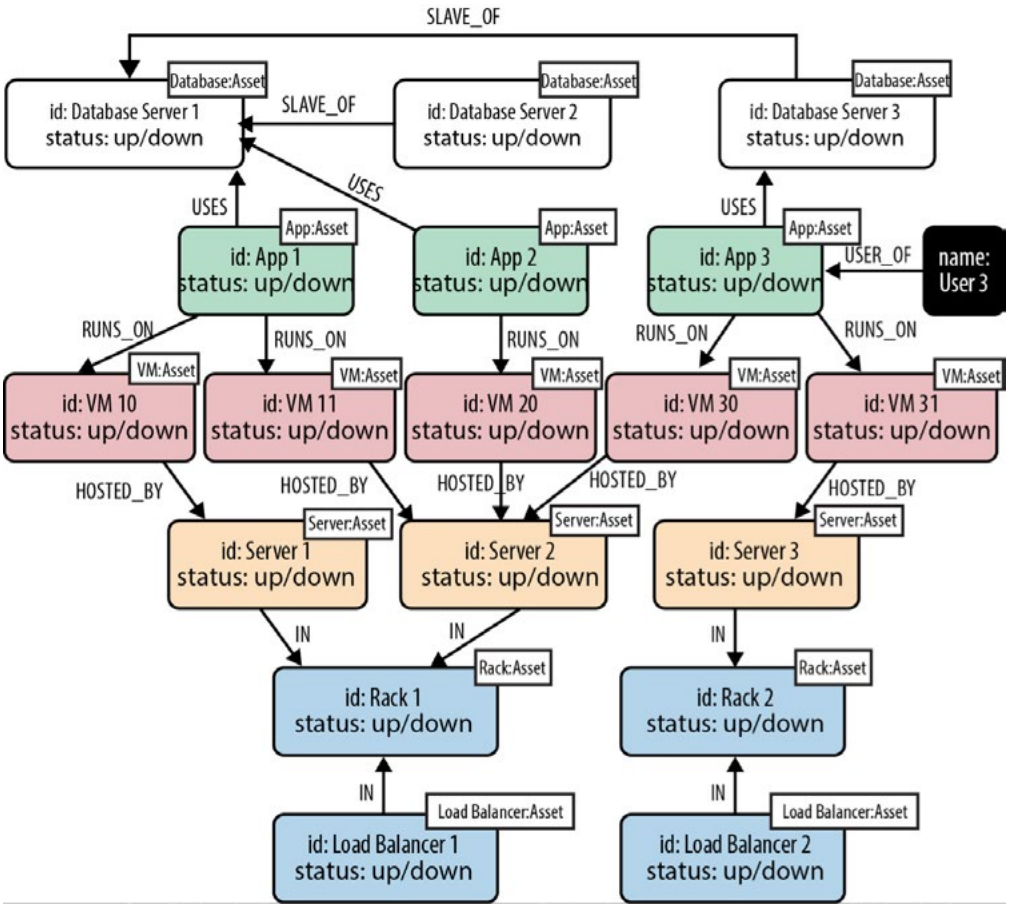

图11：关于数据中心的完整图数据模型。

需要注意的是，这里的大多数节点都有两个标签：既有特定类型的标签(如数据库、应用程序或服务器)；也有更通用的资产Asset标签。这使我们既能够针对特定类型的资产进行查询，也可以查询所有资产(不论其类型)。

与前面的关系数据库模型(在下一页会再次介绍)相比，是不是图数据模型更容易发展、包含更丰富的关系、对于业务人员而言更易于理解？

我们正是这么认为的。

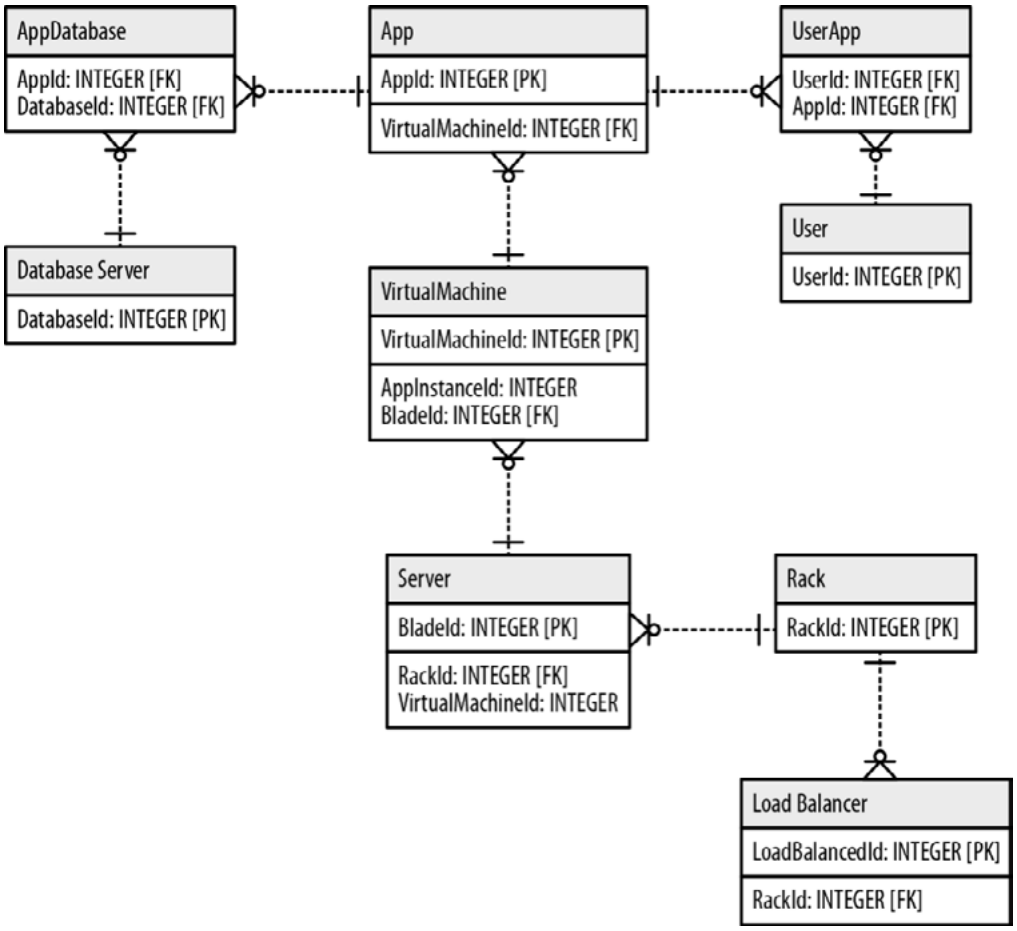

图12：数据中心领域的完整关系数据模型。 这个数据模型比我们前一页的图模型复杂得多，而且用户友好性较差。

虽然一开始就假设数据模型不会改变，会相对容易地忽略前期建模的各种头痛问题。但今天对敏捷开发实践的要求，会让你比想象中更早地再次回到白板面前讨论和应付新的需求，或者更糟糕的是，不得不求助于数据库移植专家。

结 论

需要记住的是，数据模型总是在变化。虽然一开始就假设数据模型不会改变，会相对容易地忽略前期建模的各种头痛问题。但今天对敏捷开发实践的要求，会让你比想象中更早地再次回到白板面前讨论应付新的需求，或者更糟糕的是，不得不求助于数据库移植专家。

另一方面，数据建模只是数据库开发生命周期的一部分。 能够轻松高效地查询数据(通常意味着更加实时的响应)，与拥有丰富而灵活的数据模型同样重要。在下一章中，我们将研究RDBMS和图数据库在查询语言之间的区别。

第四章、
查询语言的比较: SQL vs. Cypher

在谈到数据库查询语言时，语言效率很重要。

使用SQL查询关系数据库很容易。作为一种声明式查询语言，SQL允许在数据库工具中轻松地进行随机的查询，或者在应用程序代码中指定与用例有关的查询。即使是对象关系映射也可以使用底层的SQL与数据库通信来实现。

但是，在试图浏览**关联**的数据时，SQL遇到了主要的性能瓶颈。对于数据关联类查询问题，SQL查询语句会比实现同样功能的Cypher查询语句多出很多行(下面会进一步介绍Cypher查询语言)。

冗长的SQL查询不仅需要更多的时间来运行，而且由于其复杂性，它们也更可能包含人为的编码错误。另外，较短的查询简化了整个开发团队的理解、提高了维护的便利性。想象一下，一位来自项目外的开发人员面对一个复杂的SQL查询、并试图明白开发人员的意图，将是一件多么复杂的事情。

图查询的高效意味着它能更好地支持实时应用。在一个以单条推文(tweet)速度运行的经济体中，这种差异是不可忽视的底线。

在谈到图查询相对于SQL查询所具有的效率优势时，区别到底有多大呢？图查询比起SQL查询到底能提升多少性能？答案是：快到可以对企业运行产生足够重大的影响。

图查询的高效率意味着它能更好地支持实时应用。在一个以单条推文(tweet)速度运行的经济体中，这种差异是不可忽视的底线。

查询语言和数据模型之间的关系

值得说明的是，查询语言不仅仅是要求(即查询)数据库中的一组特定结果，它也影响数据建模结果的首要原因。

从前一章我们知道，图数据库的数据建模就像连接白板上的圆点一样简单。在白板上草拟的内容就是在数据库中存储的内容。

就其本身而言，这种建模简单性有许多实际的好处，其中最明显的是业务用户可以了解数据库开发人员实际创建的内容。更重要的是：使用正确的查询语言构建的直观模型，可确保**构建**数据模型的方式与**分析**方式之间不存在任何不匹配。

查询语言真实反应其基于的数据模型。这就是为什么SQL是关于表和连接、而Cypher是关于实体之间的关系的原因。图模型有多贴近自然，Cypher查询语言也是如此。任何领域专家，无论是技术还是非技术的，都能理解包含了由箭头相连的圆圈的图所表示的含义。

在关系数据库中，数据建模过程到目前为止已经从实际的日常SQL查询中抽象出来，这造成分析和实现之间存在很大差异。换句话说，构建关系数据库模型的过程使得从同一模型中既能有效表示数据，同时高效查询数据变得不再可能。

与此形成鲜明对比的是，图数据模型不仅清晰地表达了数据的关联方式，而且还可以就想要实现的数据查询进行清楚地沟通。对图模型而言，建模和查询只是同一枚硬币的两面。

而基于图的数据库查询语言则能够帮助我们轻松地同时应对这两面。

图查询语言Cypher简介

（注：本文不是Cypher的完整参考文档，只是其重要特性的概述。）

就像SQL是关系数据库的标准查询语言一样，Cypher是一种开放的、多供应商支持的图查询语言。openCypher项目的出现扩大了Cypher的覆盖范围，使其获得接受的程度远远超过了其最初的创造者Neo4j。

Cypher也是一种声明式查询语言，它建立在SQL的基本概念和子句之上，但增加了图特有的功能，使得在应用到含义丰富的图模型时，查询语句不会过分冗长。

Cypher旨在让开发人员、数据库专业人士和业务领域专家都能轻松阅读和理解查询逻辑。它很容易使用，因为它与我们使用图直观地描述关联数据的方式相匹配。

我们都曾尝试过编写包含大量JOIN的SQL语句，它们的存在是由于各种实现方面的限制，而这种查询会让我们很快忘记查询实际执行的操作。相比之下，由于查询结果以直观的可视化方式表达，Cypher的查询可以保持简洁并让开发人员专注于领域概念。

Cypher的基本概念是允许在数据库中查找与特定**模式**匹配的数据。通俗地说，可以要求数据库“找到像这样的东西”。而描述“类似这样的东西”的方式，使用特定的ASCII符号。

来看看下面的包含三个共同朋友的社交图：

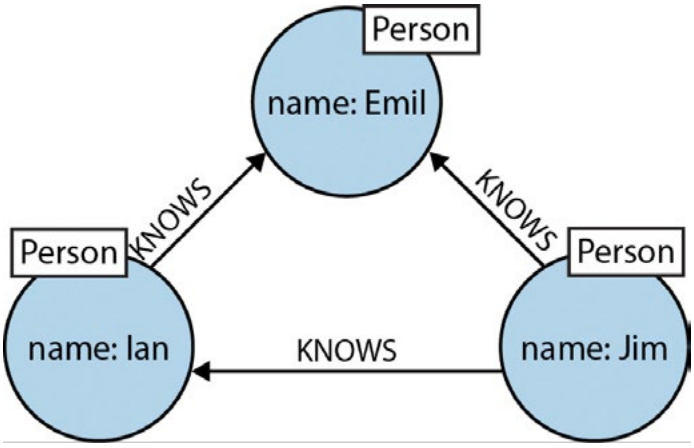

图13：描述三个朋友之间关系的社交图。

下面是使用Cypher语言来描述的关系：

```
(emil)<-[:KNOWS]-(jim)-[:KNOWS]->(ian)-[:KNOWS]->(emil)
```

这个Cypher语句描述了一条形成三角形的路径，它将一个称为Jim的节点连接到称为Ian和Emil的两个节点，并将Ian节点也连接到Emil节点。显然，Cypher描述的模型和我们在白板上绘制图的方式是一致的。

到目前为止，这一Cypher模式描述了一种简单的图结构，但它尚未涉及图数据库中的任何特定数据。要将模式绑定到现有数据集中的特定节点和关系，我们首先需要指定一些属性值和节点标签以帮助查找数据集中的相关元素。

下面是特性更丰富的Cypher语句：在这里，我们使用名称name属性和Person标签将每个节点绑定到其标识符。例如，Emil标识符绑定到数据集中的一个节点，其标签为Person，name属性的值为Emil。以这种方式将部分模式锚定到实际数据是Cypher最常见的做法。

```
(emil:Person {name:'Emil'})
     <-[:KNOWS]-(jim:Person {name:'Jim'})
     -[:KNOWS]->(ian:Person {name:'Ian'})
-[:KNOWS]->(emil)
```

使用ASCII字符来表示节点和关系，来方便地描述数据项及其之间的关系。

面向RDBMS开发人员的Cypher子句指南

像大多数查询语言一样，Cypher由子句组成。

最简单的查询由一个MATCH子句和一个RETURN子句组成。下面是一个Cypher查询的例子，使用这两个子句来查找名为Jim的用户的共同朋友(模型参见上一页的社交图)：

```
MATCH (a:Person {name:'Jim'})-[:KNOWS]->(b)-[:KNOWS]- >(
      c), (a)-[:KNOWS]->(c)
RETURN b, c
```

我们来看看每个子句的更多细节：

MATCH

MATCH子句是大多数Cypher查询的核心。使用ASCII字符来表示节点和关系，以方便地描述数据。下面的例子用圆括号表示节点：

```
(a:Person {name:'Jim'})
(b)
(c)
(a)
```

Cypher 根据所提供的关于关系和相邻节点的信息，将该模式的其余部分与紧邻该锚点的图进行匹配。

我们使用大于或小于符号（->和<-）结合破折号来表示关系，其中<和>符号表示关系的方向。 在破折号之间，关系名称用方括号括起来，并以冒号作为前缀，就像上面的查询中的例子一样：

```
-[:KNOWS]->
```

节点标签也以冒号为前缀。 正如在上面的查询中的第一个节点，Person是节点应用的标签：

```
(a:Person … )
```

然后在大括号内指定节点和关系的属性的键-值对。如下例所示：

```
( … {name:'Jim'})
```

在最早的示例查询中，我们寻找类型为Person的节点，其名称属性的值为Jim。 该查找的返回值被绑定到标识符a。 该标识符允许我们在整个查询的其余部分中引用代表Jim的节点。 理论上说，这种模式可以在整个图中出现很多次 ，特别是在大型的用户数据集中。

```
(a)-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c)
```

为了限制查询，我们需要将它的一部分锚定到图中的一个或多个位置。在指定正在寻找的一个类型为Person、名称属性值为Jim的节点时，我们已将该模式绑定到图中的特定节点，即表示Jim的节点。

然后Cypher根据所提供的关于关系和相邻节点的信息，将该模式的其余部分与紧邻该锚点的图进行匹配。 在这个过程中，它发现要绑定的其他标识符的节点。虽然a将始终与Jim绑定，但b和c在查询执行时将绑定到一系列匹配的节点。

RETURN

该子句指定对匹配的数据，哪些表达式、关系和属性应返回给客户端。在上面的示例查询中，我们感兴趣的是返回绑定到b和c标识符的节点。

其它Cypher子句

WHERE

指定过滤模式匹配结果的条件。

CREATE and CREATE UNIQUE

创建节点和关系。

MERGE

通过匹配和重复使用与提供的谓词相匹配的现有节点和关系，或通过创建新的节点和关系，确保所匹配的模式唯一存在于图中。

DELETE/REMOVE

删除节点、关系和属性。

SET

设置属性值和标签。

ORDER BY

将结果排序为RETURN的一部分。

SKIP LIMIT

跳过顶部的结果并限制结果的数量。

FOREACH

对列表中的每个元素执行更新操作。

UNION

合并来自两个或更多查询的结果。

WITH

链接多个查询部分并将结果从一个转发到下一个。与Unix中的管道命令类似。

更多命令参见最新的Cypher Refcard：https://neo4j.com/docs/cypher-refcard/current/。

比较SQL与Cypher查询

我们已经对Cypher有了基本的了解，现在是时候与SQL做一个比较，看看前者在语言效率上的优势，以及后者在处理复杂连接型查询时执行效率方面的低下。

第一个例子是面向组织机构领域（来自第3章）建立的关系数据模型：

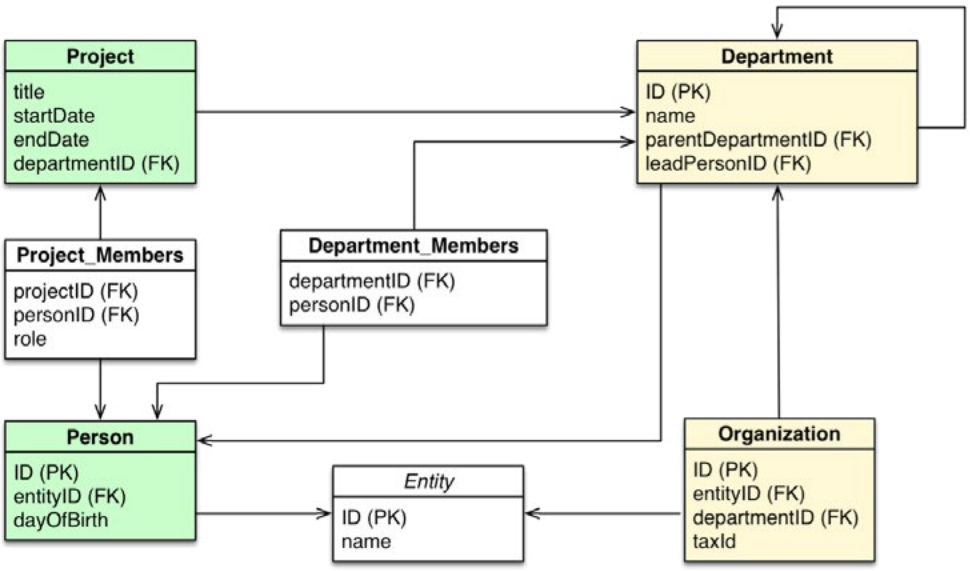

图14：组织机构的关系数据模型。

在上述描述组织机构领域的关系数据模型中，如果要列出“**IT**部门”中的员工，使用SQL语句和Cypher查询的例子分别在下面给出。

SQL Statement:

```
SELECT name FROM Person

LEFT JOIN Person_Department
  ON Person.Id = Person_Department.PersonId

LEFT JOIN Department
  ON Department.Id = Person_Department.DepartmentId

WHERE Department.name = "IT Department"
```

Cypher Statement:

```
MATCH (p:Person)<-[:EMPLOYEE]-(d:Department)

WHERE d.name = "IT Department"

RETURN p.name
```

在上一页的这个例子中，Cypher语句是SQL语句长度的一半、并且简单得多。 这个Cypher查询不仅可以更快地创建和运行，而且还可以减少出错的几率。 下面再看一个极端一些的例子。 我们将从Cypher查询开始：

Cypher Statement:

```
MATCH (u:Customer {customer_id:'customer-one'})-
[:BOUGHT]->(p:Product)<- [:BOUGHT]-(peer:Customer)-
[:BOUGHT]->(reco:Product)

WHERE not (u)-[:BOUGHT]->(reco)

RETURN reco as Recommendation, count(*) as Frequency

ORDER BY Frequency DESC LIMIT 5;
```

这个Cypher 查询实现对于每位购买产品的客户，查看其他购买了相同产品的客户还买了什么，并将其作为推荐产品返回。 WHERE 子句过滤掉客户已购买的产品，因为我们不想推荐客户他们已经购买过的产品。 Cypher查询中，MATCH子句中的每个箭头表示一种关系，该关系在关系模型中建模为多对多JOIN表，并且每个JOIN表有两个JOIN。 所以像这样简单的查询也包含了六个跨表的JOIN。 以下是用SQL写的等效查询：

```
SELECT product.product_name as Recommendation, count(1) as Frequency
FROM product, customer_product_mapping, (SELECT cpm3.product_id,
cpm3.customer_id
      FROM Customer_product_mapping cpm, Customer_product_mapping cpm2
      , Customer_product_mapping cpm3
      WHERE cpm.customer_id = 'customer-one'
      and cpm.product_id = cpm2.product_id
      and cpm2.customer_id != 'customer-one'
      and cpm3.customer_id = cpm2.customer_id
      and cpm3.product_id not in (select distinct product_id
      FROM Customer_product_mapping cpm
      WHERE cpm.customer_id = 'customer-one')
      ) recommended_products
WHERE customer_product_mapping.product_id = product.product_id
and customer_product_mapping.product_id in recommended_products.product_id
and customer_product_mapping.customer_id = recommended_products.customer_
id GROUP BY product.product_name
ORDER BY Frequency desc
```

该SQL语句的长度是等效的Cypher的三倍！而且由于JOIN的复杂性，它不仅会遇到性能问题，而且随着数据集的增长，性能也会迅速下降。

结 论

如果应用程序的性能是优先要考虑的因素，那么数据库查询语言很关键。

SQL针对关系数据库模型进行了充分优化，然而当它必须处理复杂的、面向关系的查询时，其性能会迅速下降。在这些情况下，根本问题不在于SQL、而在于关系模型本身不适用于处理连接的数据。

在数据高度关联的应用领域，图模型是必须的，使用像Cypher这样的图查询语言也是必须的。 如果开发团队有来自SQL的背景，那么学习Cypher其实很容易、执行查询则更容易。

在规划下一个数据驱动的、企业级的应用时，确保支持它的查询语言都是为了提高速度和效率而构建的，那你一定会对最终的结果非常满意。

在数据高度关联的应用领域，图模型是必须的，使用像Cypher这样的图查询语言也是必须的。

第五章、
从RDBMS导入数据到Graph

无论你准备将整个传统RDBMS移植到图数据库中，还是正在同步数据库以实现多类型数据存储(Polyglot)，或者只是进行初步的概念和技术验证，很多时候都需要将数据从关系数据库导入到图数据库中。在本章中，我们将展示如何使这个过程尽可能平滑和无缝。

第一步是确保对图数据模型（即节点、关系、标签、属性和关系类型)有正确的理解，特别是它们如何适用于给定的问题领域。

事实上，在开始数据导入之前，至少应该在白板上完成基本图模型，提前了解你的数据模型会使导入过程轻松很多。

将数据从关系数据库移植到图数据库有三种主要方法。哪种方法最适合应用或IT架构取决于要达到的特定目标。

导入RDBMS数据的策略性方法

将数据从关系数据库移植到图数据库有三种主要方法。 哪种方法最适合应用或IT架构取决于你的特定目标。

下面，可以看到在关系数据库和图数据库之间处理数据的每种方法：

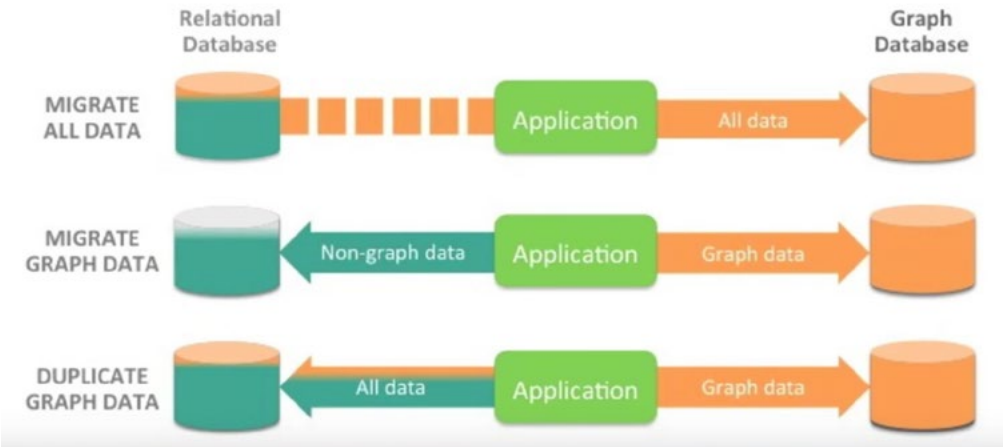

图15：在关系数据库和图数据库之间处理数据的三种最常见的方法。

第一种方法，开发团队将他们的所有数据从旧的关系数据库迁移到图数据库中。 这通常是一次性批量迁移。

第二种方法，开发人员将其关系数据库继续用于任何依赖非图的、表格数据的用例。 然后，对于涉及大量JOIN或数据关系的用例，将这些数据存储在图数据库中。

第三种方法，开发团队将其所有数据同时复制到关系数据库和图数据库中。这样，可以以任何形式查询数据，使其最适合试图运行的各类查询。

在管理RDBMS和图数据方面，没有“绝对正确”的方法。你的团队应该考虑应用的目标、最频繁的应用案例和最常见的查询，并为特定的IT环境选择最合适的解决方案。

从RDBMS中导出数据

如果决定需要将关系数据库中的数据导入到图数据库中，那么首先要从现有的RDBMS中提取数据。

大多数关系数据库都允许转储整个表或整个数据集，或者将数据存储到CSV文件，或者发送查询给数据库。这些任务通常只是数据库本身的复制功能。当然，在许多情况下，导出的数据文件会驻留在数据库服务器上，需要从那里下载它，这可能反而是更大的一个挑战。

另一种选择是使用数据库驱动程序（如JDBC或其他驱动程序）访问关系数据库，以提取需要的数据集。

此外，如果想在关系数据库和图数据库之间建立同步机制，那么定期根据时间戳或其他更新的标志来读取给定表以实现数据同步也是常用的方法。

另一个需要考虑的方面是许多关系数据库并不是为在短时间内输出大量数据而优化的。因此，如果你试图将数据直接从RDBMS迁移到图数据库，该过程很可能会慢得无法忍受。

例如，在一个实际案例中，一个Neo4j客户在MySQL集群中存储大型社交网络的数据。从MySQL数据库导出数据需要三天时间，而将数据导入Neo4j只需要三个小时。

准备好开始导入数据了？这里有最后一个提示：在写入磁盘时，请确保禁用病毒扫描程序并检查磁盘时间表，以便尽可能获得最高的磁盘性能。在导入过程中，检查或设置可能会提高性能的任何其他选项也是值得的。

在一个实际案例中，一个Neo4j客户在MySQL集群中存储大型社交网络的数据。从MySQL数据库导出数据需要三天时间，而将数据导入Neo4j只用了三个小时。

使用LOAD CSV导入数据

导入来自关系数据库的数据的最简单方法，是创建从单个实体表和JOIN表导出的CSV数据文件。CSV格式是各种不同应用程序之间数据格式的最常用(也是最低)的共同标准。虽然CSV格式本身不怎么受欢迎，但在将数据导入图数据库时，它是最容易使用的。

在Neo4j中，LOAD CSV是一个Cypher命令，它允许你将来自HTTP或文件URL的CSV文件加载到数据库中。每行数据都在Cypher语句中访问和处理，然后从这些行中实际创建或更新图中的节点和关系。

LOAD CSV命令是将平面数据（即CSV文件）转换为连接的图数据的强大方法。LOAD CSV既可用于单表导出的CSV文件，也可用于包含完全扁平化的表，或者包含多个表JOIN的数据导出文件。

LOAD CSV允许在导入过程中转换、过滤或解构导入数据。还可以使用此命令分割区域，提取单个值或遍历特定属性列表，然后将其过滤提取为单个属性。

最后，在LOAD CSV中还可以控制事务的大小，因此不会在特定命令中遇到内存不足的问题。另外，可以通过 Neo4j shell工具（而不仅仅是Neo4j浏览器）来运行LOAD CSV，这使得更容易执行脚本实现数据导入。

总结一下，Cypher的LOAD CSV命令可以用来：

- 接收数据，通过字段列名称或偏移量访问列
- 将字符串中的值转换为不同的格式和结构（浮动，拆分...）
- 跳过要忽略的行
- MATCH，基于属性查找匹配节点
- CREATE创建或MERGE合并节点或关系，根据源数据行中的标签和属性
- 为节点设置新的标签和属性，或删除过时的标签和属性

LOAD CSV的一个例子

以下是使用LOAD CSV将CSV文件导入Neo4j的简要示例。

Example file: persons.csv

```
name;email;dept
"Lars Higgs";"lars@higgs.com";"IT-Department"
"Maura Wilson";"maura@wilson.com";"Procurement"
```

Cypher Statement:

```
LOAD CSV FROM ' ile:///data/persons.csv' WITH HEADERS AS line FIELDTERMINATOR ";"
MERGE (person:Person {email: line.email}) ON CREATE SET person.name = line.name
MATCH (dep:Department {name:line.dept})
CREATE (person)-[:EMPLOYEE]->(dept)
```

你可以从一个或多个数据源（包括RDBMS）中导入多个CSV文件，以丰富领域模型的数据，增加数据能提供的相关的洞察力和功能。

除了LOAD CSV，还有其他专用导入工具可以帮助高效导入更大容量(超过千万行)的数据，下面我们会继续介绍。

命令行的批量导入工具

neo4j-admin import命令是用于大批量数据插入的、伸缩性强的导入工具。该工具读取CSV文件，将插入操作扩展到所有可用的CPU和磁盘容量中，将数据暂存在中间存储架构中，并在可能情况下并行执行所有导入任 务。

然后，该工具对以中间格式暂存的数据使用特定的先进内存压缩技术来创建新的图结构。

命令行批量加载工具执行效率非常高：每秒能够导入一百万条记录，处理包含数十亿个节点、关系和属性的大型数据集。neo4j-admin import命令对初始化数据库、一次性导入大量数据特别有用。

客户化基于Cypher的数据加载工具

为了导入数据，还可以使用各种Neo4j API自动运行特定的Cypher语句。这些API可以使用Cypher的运行、创建、更新和合并命令。

一种API是HTTP End-Point，在所有驱动程序中都支持，可以直接从开发语言的HTTP客户端或HTTP库中访问HTTP End-Point。

使用HTTP，可以将数据从关系数据库（或其他数据源）中提取出来，并将其转换为Cypher语句需要的格式和参数，然后在那之上批量执行，并控制导入事务。

从Neo4j 2.2开始，基于Cypher的、客户定制的加载程序在高度并发写入的情况下也可以很好地执行。在一次测试中，使用高度并发的Cypher语句，实现了每秒插入100万个节点和关系。

基于Cypher的加载方法可以与许多不同的驱动程序一起使用，包括JDBC驱动程序。如果已经有一个使用JDBC的ETL工具或Java程序，则可以使用Neo4j的JDBC驱动程序将数据导入Neo4j。因为Cypher语句是可以用字符串表示的查询语句，在这种情况下，还可以为Cypher语句提供参数。

在一次测试中，使用高度并发的Cypher语句，实现了每秒插入100万个节点和关系。

其他RDBMS到图数据库的数据导入资源

本章仅概要介绍了从关系数据库中将数据导入图数据库的三种最常用的方法 。以下是关于数据导入的其他方法的更多资源，以及关于上述三种方法的更深入的指导：

- Guide: Data Import
- Manual: LOAD CSV
- Webinar: Data Import
- Guide: CSV Import
- Tool: Direct RDBMS Import
- Tool: SQL to Neo4j Import
- Blog: Importing AdventureWorks Data into Neo4j
- Neo4j for Relational MetaData (SQLServer)

这本书仅仅针对现今的RDBMS开发人员初步介绍了图数据库。

更多资源

适用于RDBMS开发者的关于图数据库的更多信息

这本书仅仅针对现今的RDBMS开发人员初步介绍了图数据库。要获得关于关系数据库和图数据库的比较，以及更多观点、技巧和窍门，请访问以下资源：

- 《图数据库权威指南》, 张帜 主编，清华大学出版社，2017
- 《Neo4j 3.X入门经典》, 张帜 主编, 即将出版
- Neo4j中文社区：http://neo4j.com.cn/
- Neo4j 中文社区 技术交流 QQ群： 547190638
- The Graph Databases for Beginners blog series
- Developer Guide: From Relational to Neo4j
- O'Reilly's ***Graph Databases 2nd Edition*** ebook
- ***Learning Neo4j*** ebook
- Presentation: From Relational to (Big) Graph
- Webinar: From RDBMS to Graphs
- Webinar: Relational to Graph: Data Modeling
- Webinar: Relational to Graph: Importing Data into Neo4j
- Online Training: Getting Started with Neo4j
- Classroom Trainings

关于Neo4j:

Neo4j是全球图数据库技术的领导者。作为全球部署最为广泛的图数据库，我们帮助了包括：Comcast（康卡斯特）、NASA（美国国家航空航天局）、UBS（瑞银）及Volvo Cars（沃尔沃）在内的国际品牌预测并揭示了人、流程、系统之间如何进行关联和联系的模式和结构。使用这种以关系优先的方法，基于Neo4j构建的应用程序可以解决大数据带来的挑战，例如人工智能与分析、欺诈检测、实时推荐及知识图谱。 更多详情，请见neo4j.com官网。

For more information on Neo4j, contact us via email or phone:
1-855-636-4532
apac@neo4j.com