neo4j

O banco de dados orientados à grafos mais utilizado no mundo

EBOOK

Guia Definitivo de Bancos de Dados Grafos:

Para Desenvolvedores RDBMS

Por Michael Hunger, Ryan Boyd e William Lyon

Ebook

ÍNDICE

Guia Definitivo de Bancos de Dados Grafos:
Para Desenvolvedores RDBMS

Por Michael Hunger, Ryan Boyd e William Lyon

Introdução:
Porque nós escrevemos este ebook

Quando duas tecnologias de banco de dados compartilham espaço no mesmo título do livro, é provável que haja confusão quanto aos motivos de seu conteúdo.

Primeiras coisas primeiro: Não escrevemos este livro para criticar bases de dados relacionais ou para criticar esta tecnologia valiosa. Sem bancos de dados relacionais, muitos dos mais cruciais sistemas e aplicativos não seriam executados e, sem a inovação inicial dos pioneiros do RDBMS, nunca chegaríamos onde estamos com a atual tecnologia de banco de dados.

Em vez disso, escrevemos este livro para apresentá-lo - um desenvolvedor com experiência em RDBMS - a uma tecnologia de banco de dados que mudou não só como vemos o mundo, mas como construímos o futuro. Os requisitos atuais de negócios e usuários exigem aplicativos que conectam mais e mais dados do mundo, mas ainda esperam altos níveis de desempenho e confiabilidade de dados.

Acreditamos que essas aplicações do futuro serão construídas utilizando bancos de dados orientado à grafos e nós não queremos que você seja deixado para trás. Na verdade, estamos aqui para ajudá-lo em todas as etapas do processo de aprendizagem.

Enquanto outros bancos de dados NoSQL (não apenas SQL) anunciam a si mesmos como rivais da tecnologia RDBMS, preferimos ser um recurso útil para ajudar você a adicionar bancos de dados grafos ao seu conjunto de habilidades profissionais.

Bancos de dados relacionais ainda têm seus casos de uso perfeitos. Mas para aqueles casos em que você precisa de uma solução diferente, esperamos que este livro ajude você a reconhecer quando - e como - usar um banco de dados grafo para enfrentar esses novos desafios.

Ao ler e examinar essas páginas, sinta-se à vontade para entrar em contado e expor as suas dúvidas. Você pode mais comumente nos encontrar em Stack Overflow, nosso Google Group ou no nosso canal Slack.

Boa leitura!

Michael, Ryan, e Will

Capítulo 1:
Porque bancos de dados relacionais não são sempre suficientes

Bancos de dados relacionais são ferramentas poderosas.

Desde os anos 80, foram a escolha da maioria das aplicações de software e continuam a ser assim atualmente. Bancos de dados relacionais (RDBMS) foram inicialmente projetados para codificar estruturas tabulares e formulários, e eles fazem isso muito bem. Para o uso correto e a arquitetura certa, eles são uma das melhores ferramentas para armazenar e organizar dados.

Porque bancos de dados relacionais armazenam dados altamente estruturados em tabelas com colunas predeterminadas e muitas linhas do mesmo tipo de informação, eles exigem dos desenvolvedores, estruturar os dados previamente que serão utilizados em aplicações.

Mas os requisitos do usuário e aplicações atuais estão pedindo por mais. Mais recursos, mais dados, mais agilidade, mais velocidade e - mais importante - mais conexões.

A Incompatibilidade Entre Bancos de Dados Relacionais e Relacionamentos Entre Dados

Apesar de seu nome, bancos de dados relacionais não são adequados para dados altamente conectados que possuímos hoje, porque eles não armazenam robustamente relacionamentos entre elementos de dados.

[É interessante notar que bancos de dados relacionais são nomeados em consequência da notação matemática, já que são altamente específicos como parte da álgebra relacional. O nome em si não é derivado para descrever relações entre dados.]

Tradicionalmente, os desenvolvedores foram ensinados a armazenar dados em colunas e linhas de um modelo relacional. Porém, linhas e colunas não são realmente como dados existem no mundo real. Em vez disso, dados existem como objetos e relações entre os diferentes tipos de objetos.

Esses tipos de dados complexos, reais, estão aumentando em volume, velocidade e variedade. Como resultado, as relações de dados – que são muitas vezes mais valiosas do que os dados em si – estão crescendo em um ritmo ainda mais rápido.

O problema: Bancos de dados relacionais não são projetados para capturar essas ricas informações de relacionamento.

A Ágil Realidade das Aplicações de Software Atuais

Todas as equipes de desenvolvimento enfrentam a realidade de constante mudança de negócios e requisitos do usuário realizando modificações frequentes e ajustes de uma determinada arquitetura de dados.

Administradores de banco de dados (DBAs) e desenvolvedores enfrentam um fluxo constante de solicitações de negócios para adicionar elementos ou atributos para atender novas exigências – tais como o armazenamento de informações sobre a mais recente plataforma social – mas essas alterações de esquema regular são problemáticas para desenvolvedores RDBMS e vêm com uma manutenção de alto custo.

Isto acontece porque os bancos de dados relacionais não se adaptam bem em cenários de frequentes mudanças. Em vez disso, seu esquema fixo funciona melhor para problemas que são bem definidos desde o início.

O redesenho de esquema é lento e caro, também prejudica o processo de desenvolvimento de software ágil, impedindo a capacidade de sua equipe de inovar rapidamente

- um custo de oportunidade significativo, independentemente do tamanho do negócio e/ou equipe.

O veredito: Bancos de dados relacionais não são projetados para a velocidade da agilidade do negócio.

Como as Consultas de Dados Conectados Controlam o Desempenho do RDBMS

Apesar dos avanços em computação, processadores mais rápidos e redes de alta velocidade, o desempenho de algumas aplicações de banco de dados relacionais continua a diminuir. Esta queda de desempenho tem vários sintomas conhecidos (consulte a seção abaixo), mas a causa raiz normalmente se resume a um fator: consultas utilizando inúmeras conexões entre tabelas.

Uma vez que os bancos de dados relacionais não são criados ou otimizados para lidar com dados conectados, qualquer tentativa de responder a consultas de relacionamento de dados - como um mecanismo de recomendação, um padrão de detecção de fraude ou um grafo social - envolve inúmeros JOINs entre tabelas de banco de dados.

Em bancos de dados relacionais, as referências à outras linhas e tabelas são indicadas referindo-se aos atributos de chave primária por meio de colunas de chave estrangeira. (Veja a Figura 1 abaixo para um exemplo.)

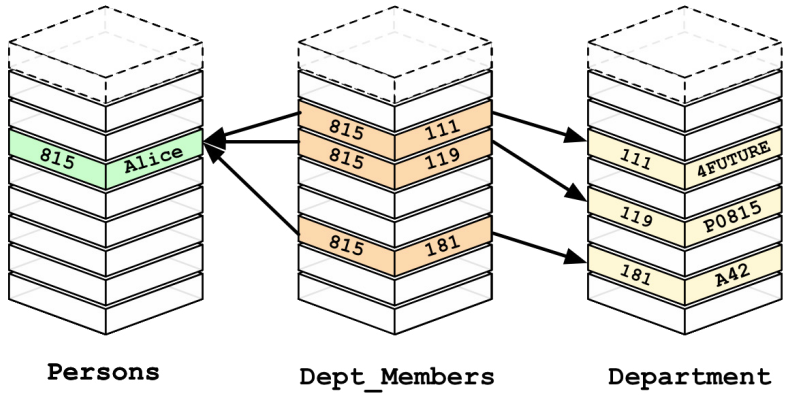

Figura 1: uma tabela de membros de departamento e seus JOINs entre as tabelas Pessoas e Departamentos em um banco de dados relacional utilizando chave estrangeiras.

Essas referências são construídas com constraints, mas somente quando a referência nunca é opcional. Os JOINs são então computados no tempo de consulta combinando chaves primárias e estrangeiras das muitas linhas (potencialmente indexadas) das tabelas à serem "juntadas". Tais operações são intensivas em processamento e em memória com um custo exponencial à medida que as consultas crescem.

Consequentemente, modelar e armazenar dados conectados torna-se impossível sem extrema complexidade. Essa complexidade funciona em casos como instruções SQL que requerem dezenas de linhas de código apenas para realizar operações simples. O desempenho geral também se caracteriza pela complexidade da consulta, o número e os níveis de relações de dados e o tamanho geral do banco de dados.

Com o dia a dia cada vez mais acelerado e tudo em tempo real, expectativas de aplicações de software que respondam de maneira instantânea, os bancos de dados relacionais tradicionais são simplesmente inadequados sempre que as relações de dados são fundamentais para o sucesso.

5 Sinais que Seu Aplicativo RDBMS Sofre com Alto Esforço SQL

Muitas aplicações de banco de dados relacionais estão funcionando bem dentro dos seus limites. Algumas, no entanto, podem apresentar sinais significativos de tensão induzida pelo banco de dados, especialmente quando um RDBMS está sendo usado para lidar com dados altamente conectados.

Aqui estão cinco dos sinais mais comuns que você pode tentar resolver um problema de dados conectados com um banco de dados relacional:

1. Um grande número de JOINs

Quando você utiliza consultas que realizam um número elevado de JOINs, há uma explosão de complexidade e consumo de recursos de computação. Isso resulta em um aumento correspondente nos tempos de resposta da consulta.

2. Numerosos Self-JOINs (ou JOINs Recursivos)

As declarações de auto-associação ou self-joins são comuns para a hierarquia e as representações de árvores de dados, mas as relações cruzadas, por meio da associação repetida de tabelas a elas mesmas, são ineficientes. De fato, algumas das mais longas consultas SQL no mundo envolvem JOINs recursivas.

3. Mudanças frequentes no esquema

Numa altura em que a agilidade do negócio é superior, os pedidos de alterações são mais frequentemente descartados por DBAs porque o esquema de bancos de dados relacionais não foi projetado para modificações frequentes. As mudanças comuns no esquema indicam que os dados ou requisitos estão evoluindo rapidamente, exigindo um modelo de dados mais flexível.

4. Consultas de execução lenta (apesar do ajuste extensivo)

Seu DBA pode usar cada truque no livro ou manual para acelerar os tempos de consulta, mas muitas consultas SQL ainda não são rápidas o suficiente para suportar as necessidades do seu aplicativo. Além disso, desnormalizando os modelos de dados para obter desempenho, pode afetar negativamente a qualidade dos dados armazenados.

5. Pré-computação de seus resultados

Como as consultas funcionam tão devagar, muitos aplicativos pré-computam seus resultados usando dados passados. No entanto, isso está efetivamente usando os dados ultrapassados para consultas que devem ser tratadas em tempo real. Além disso, seu sistema geralmente deve pré-calcular 100% de seus dados, mesmo que apenas 1-2% dele seja acessado em qualquer momento.

Uma Alternativa Aos Bancos de Dados Relacionais

Conforme mencionado anteriormente, as bases de dados relacionais têm seus casos de uso apropriados. Para esquemas altamente estruturados e predeterminados, um RDBMS é a ferramenta perfeita.

Mas, como já vimos, os bancos de dados relacionais nem sempre são suficientes. As aplicações que requerem informações de dados conectadas não podem confiar no modelo relacional.

O volume, a velocidade e a variedade dos dados de hoje - e as relações de dados - requerem uma solução que seja projetada desde o início para armazenar e organizar dados conectados. Então é hora de sabermos mais sobre bancos de dados orientados à grafos.

Capítulo 2:
Por Que Bancos de Dados Orientados à Grafos?

Nós já sabemos que os bancos de dados relacionais não são suficientes para lidar com o volume, velocidade e variedade dos dados de hoje, mas qual é a alternativa clara?

Há muitas outras opções de banco de dados - incluindo uma série bancos de dados NoSQL -, mas nenhuma delas é explicitamente projetada para lidar e armazenar relacionamentos de dados. Isto é, exceto os bancos de dados grafos.

O maior valor que os grafos trazem para a stack de desenvolvimento é a capacidade de armazenar relacionamentos e conexões como entidades de primeira classe.

Por exemplo, os primeiros adotantes da tecnologia de grafos reimaginaram seus negócios em torno do valor das relações de dados. Essas empresas tornaram-se líderes da indústria: LinkedIn, Google, Facebook e PayPal.

Como pioneiros em tecnologia, cada uma dessas empresas teve que criar seu próprio banco de dados grafo a partir do zero. Felizmente para os desenvolvedores de hoje, isso não é mais o caso, já que a tecnologia de banco de dados grafos agora está disponível no mercado e com fácil acesso.

Vamos dar uma olhada em porque você deve considerar um banco de dados grafo para sua próxima aplicação de dados conectados. Começaremos com algumas definições básicas.

O Que é Um Grafo? O Que é um Banco de Dados Grafo?

Você não precisa entender a magia matemática arcana da teoria dos grafos para entender os bancos de dados de grafos. Pelo contrário, se você já está familiarizado com bancos de dados relacionais, você se familiziará com o modelo de grafos.

Primeiro: um grafo/gráfico - em matemática - não é o mesmo que um grafo, então não imagine um gráfico de barras ou linhas.

Em vez disso, imagine uma rede ou mapa mental, como no exemplo abaixo.

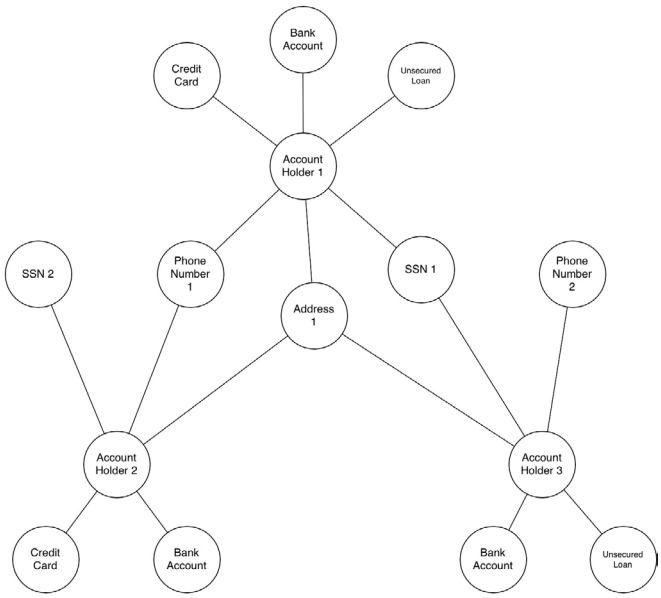

Figura 2: Um grafo básico de um círculo de fraude que compartilha informações de contato semelhantes.

Um grafo é composto de dois elementos: um nó e um relacionamento.

Cada nó representa uma entidade (uma pessoa, um lugar, uma coisa, uma categoria ou outra peça de dados) e cada relação representa como dois nós estão associados. Por exemplo, dois nós, como: "bolo" e "sobremesa" teriam o relacionamento "é um tipo de" apontando de "bolo" para "sobremesa". Logo que bolo pode ser um tipo de sobremesa

Esta estrutura de propósito geral permite modelar todos os tipos de cenários - de um sistema de estradas, a uma rede de dispositivos, ao histórico médico de uma população ou qualquer outra coisa definida pelos relacionamentos.

O Que é um Banco de Dados Grafo?

Um banco de dados orientado à grafo é um sistema de gerenciamento de banco de dados on-line com as operações Criar, Ler, Atualizar e Excluir (CRUD) trabalhando em um modelo de dados baseado em grafos. Os bancos de dados orientados à grafos geralmente são construídos para uso com sistemas de transação (OLTP). Conseqüentemente, eles normalmente são otimizados para o desempenho transacional e projetado com a integridade transacional e a disponibilidade operacional em mente.

Ao contrário de outros bancos de dados, os relacionamentos ocupam a primeira prioridade em bancos de dados grafos. Isso significa que seu aplicativo não precisa inferir conexões de dados usando chaves estrangeiras ou processamento lógico em tempo de execução, como MapReduce.

Ao montar as abstrações simples de nós e relacionamentos em estruturas conectadas, os bancos de dados de grafos nos permitem construir modelos sofisticados que mapeiem de perto o nosso domínio problemático.

Existem duas propriedades importantes das tecnologias de banco de dados de grafos:

Armazenamento em Grafos

Alguns bancos de dados orientados a grafos usam armazenamento de grafo nativo que é projetado especificamente para armazenar e gerenciar grafos, enquanto outros usam bancos de dados relacionais ou orientados a objetos. O armazenamento não-nativo é muitas vezes muito mais latente, especialmente à medida que o volume de dados e a complexidade da consulta crescem.

Motor de Processamento de Grafos

O processamento de grafos nativo (a.k.a. "adjacência livre de índice") é o meio mais eficiente de processar dados de grafos porque os nós conectados fisicamente "se posicionam" junto um ao outro no banco de dados.

O processamento de grafo não nativo usa outros meios para processar operações CRUD que não são otimizadas para grafos.

Quais São as Vantagens de Usar um Banco de Dados Orientados a Grafo?

Um banco de dados orientados a grafo é projetado para lidar com dados altamente conectados e o aumento do volume e da conexão dos dados de hoje apresenta uma tremenda oportunidade para uma vantagem competitiva sustentável.

Quando se trata de aplicar um banco de dados orientados a grafo a um problema do mundo real, com restrições técnicas e de negócios do mundo real, as organizações empresariais escolhem os bancos de dados orientados a grafos pelas seguintes razões:

Desempenho de Minutos a Milissegundos

O desempenho e a capacidade de resposta da consulta estão no topo das preocupações de muitas organizações em relação às suas plataformas de dados. Os sistemas de transações on-line - aplicações web em grande escala - devem responder aos usuários finais em milissegundos se quiserem ter sucesso. No mundo relacional, à medida que o tamanho do conjunto de dados da aplicação cresce, o custo de se utilizar JOINs começam a se manifestar e o desempenho deteriora-se. Usando a adjacência livre de índice, um banco de dados orientados a grafo transforma JOINs complexos em curvas de grafos rápidos, mantendo o desempenho de milissegundos independentemente do tamanho geral do conjunto de dados.

Ciclos de Desenvolvimento Drasticamente Acelerados

O modelo de dados orientados a grafos reduz a incompatibilidade de impedância que tem afetado o desenvolvimento de software por décadas, reduzindo assim a sobrecarga de desenvolvimento de traduzir de um lado para o outro entre um modelo de objeto e um modelo relacional tabular.

Mais importante ainda, o modelo de grafo reduz a incompatibilidade de impedância entre os domínios técnico e comercial. Especialistas em assuntos, arquitetos e desenvolvedores podem conversar sobre o domínio principal usando o modelo compartilhadoe o modelo grafo é incorporado diretamente no aplicativo, reduzindo assim, tempo de desenvolvimento e/ou planejamento.

Capacidade de Resposta Extrema ao Negócio

As aplicações bem-sucedidas raramente ficam inertes ao tempo. Mudanças nas condições de negócios, comportamentos de usuários e infraestruturas técnicas e operacionais exigem novos requisitos. No passado, isso exigiu que as organizações realizassem cuidadas e longas migrações de dados que envolvessem modificar esquemas, transformar dados e manter dados redundantes para servir recursos antigos e novos.

O desenvolvimento com bancos de dados orientados a grafos alinha perfeitamente com as atuais práticas de desenvolvimento orientadas para testes atuais, permitindo que seu banco de dados orientados a grafo evolua em conjunto com o resto do aplicativo e com todos os requisitos comerciais em constante mudança. Ao invés de modelar exaustivamente um esboço previamente, as equipes de dados podem adicionar à estrutura feita em grafos existente sem pôr em perigo a funcionalidade atual.

Pronto Para Empresas

Quando empregado em uma aplicação de missão crítica, uma tecnologia de dados deve ser robusta, escalável e, na maioria das vezes, transacional. Embora alguns bancos de dados orientados a grafos sejam bastante novos e ainda não totalmente maduros, existem bancos de dados grafos no mercado que fornecem todas as capacidades necessárias para as grandes empresas hoje:

- Transação ACID
- Alta disponibilidade
- Escalabilidade de leitura horizontal
- Armazenamento de bilhões de entidades

Essas características têm sido um fator importante que leva à adoção de bancos de dados orientados à grafos por grandes organizações, não apenas em modestas capacidades off-line ou departamentais, mas de formas que realmente transformam o negócio.

Quais São os Casos Mais Comuns de Uso em Banco de Dados Orientados a Grafos?

Enquanto os bancos de dados de grafos se tornaram populares entre as aplicações sociais da web de consumidores (Facebook, LinkedIn, Twitter), os casos de uso se estendem muito além do espaço social.

As organizações empresariais de hoje usam tecnologia de banco de dados orientado à grafos de diversas maneiras, incluindo esses seis casos de uso mais comuns:

- Detecção de fraude
- Mecanismos de recomendação em tempo real
- Gerenciamento de dados mestre (MDM)
- Operações de rede e TI
- Gerenciamento de identidade e acesso (IAM)
- Pesquisa baseada em grafo

Para obter mais informações sobre casos de uso de tecnologia gráfica, consulte: Os 5 principais casos de uso de banco de dados orientados a grafos: desbloqueando novas possibilidades com dados conectados.

Capítulo 3: Modelagem de Dados: Modelos Relacionais vs. Grafos

Em alguns aspectos, os bancos de dados orientados à grafos são como a próxima geração de bancos de dados relacionais, mas com suporte de primeiro nível para "relacionamentos", ou as conexões implícitas indicadas através de chaves estrangeiras em bancos de dados relacionais tradicionais.

Cada nó (entidade ou atributo) no modelo de banco de dados orientado à grafos contém, de forma direta e física, uma lista de registros de relacionamento que representam suas relações com outros nós. Esses registros de relacionamento são organizados por tipo e direção e podem conter atributos adicionais.

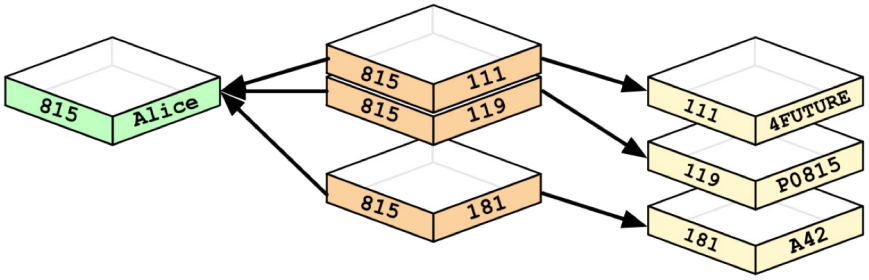

Figura 3: Um híbrido de tabela orientada à grafo / JOIN que mostra as relações de dados de chave estrangeira entre as tabelas de Pessoas e Departamentos em um banco de dados relacional.

Sempre que você executa o equivalente a uma operação JOIN, o banco de dados apenas usa essa lista e tem acesso direto aos nós conectados, eliminando a necessidade de uma computação de pesquisa e correspondência custosa.

Essa capacidade de pré-materializar relacionamentos em estruturas de banco de dados permite que bancos de dados de grafos como o Neo4j ofereçam uma vantagem de desempenho de consulta, que vão de minutos a milissegundos, especialmente para consultas com alto número de JOINs.

Os modelos de dados resultantes são muito mais simples e, ao mesmo tempo, mais expressivos que os produzidos usando bases de dados relacionais tradicionais ou outros bancos de dados NoSQL.

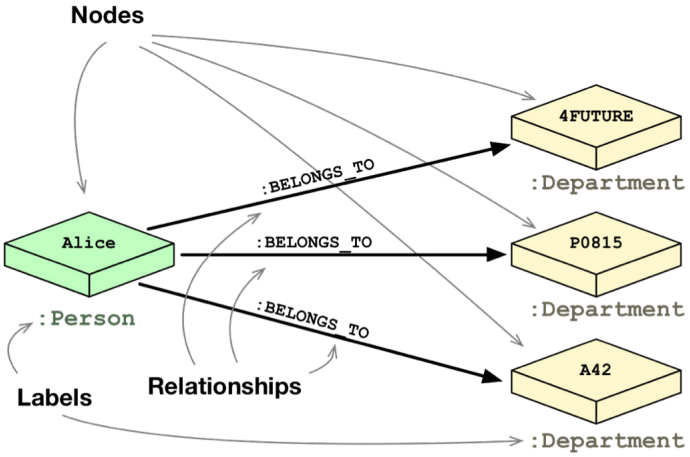

Figura 4: Um modelo de dados orientados a grafos de nossos dados originais de Pessoas e Departamentos. Nós e relacionamentos substituíram nossas tabelas, chaves estrangeiras e tabela JOIN.

Os bancos de dados orientados à grafos suportam um modelo de dados muito flexível e simples que permite modelar e gerenciar domínios ricos de forma fácil e intuitiva.

Você mais ou menos mantém os dados como está no mundo real: entidades pequenas, normalizadas, mas ricamente conectadas. Isso permite que você consulte e visualize seus dados a partir de qualquer ponto de interesse imaginável, suportando muitos casos de uso diferentes (consulte o Capítulo 2 para obter mais informações).

O modelo com maior granulação também significa que não existe um limite fixo em torno dos agregados, de modo que o escopo das operações de atualização é fornecido pelo aplicativo durante a operação de leitura ou gravação. As transações agrupam um conjunto de atualizações de nó e relacionamento em uma operação Atômica, Consistente, Isolada e Durável (ACID).

Os bancos de dados orientados a grafos como o Neo4j suportam totalmente esses conceitos transacionais, incluindo registros de gravação contínua e recuperação após término anormal, de modo que você nunca perca os dados que foram comprometidos com o banco de dados.

Se você é experiente em modelagem com bancos de dados relacionais, pense na facilidade e beleza de um diagrama de relacionamento entre entidades bem-feito e normalizado: um modelo simples e fácil de entender, que você pode redirecionar rapidamente com seus colegas e especialistas do domínio. Um grafo é exatamente isso: um modelo claro do domínio, focado nos casos de uso que você deseja suportar de forma eficiente.

Vamos tomar um modelo do domínio organizacional e mostrar como ele seria modelado em um banco de dados relacional versus o banco de dados orientados a grafos.

Primeiro, nosso modelo de banco de dados relacional:

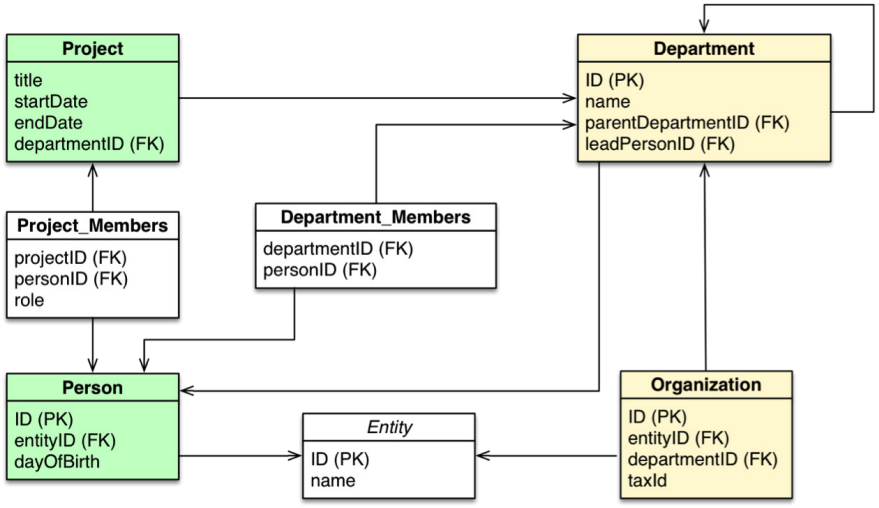

Figura 5: Um modelo de banco de dados relacional de um domínio com pessoas (Person) e projetos (Project) com uma organização (Organization) com vários departamentos (Department).

Se adequássemos este modelo de banco de dados relacional (acima) a um modelo de banco de dados orientado à grafos, passaremos pela seguinte lista de verificação para ajudar na transformação:

- Cada linha em uma tabela de entidade é um nó.
- Cada tabela de entidade é representada por uma label com nós inclusos.
- As colunas dessas tabelas se tornam propriedades do nó.
- Remova chaves primárias técnicas, mas mantenha as chaves primárias de negócios.
- Adicione constraints únicas para chaves primárias de negócios e adicione índices para atributos de pesquisa frequentes.

- Substitua chaves estrangeiras por relações com a outra tabela, remova-as depois se necessário.As colunas dessas tabelas se tornam propriedades do nó.
- Remover dados com valores padrão, sem necessidade de armazenar aqueles dados não utilizados.
- Os dados nas tabelas que são desnormalizados e duplicados podem ter que ser retirados em nós separados para obter um modelo mais limpo.
- Os nomes das colunas indexadas podem indicar uma propriedade de arrays (como email1, email2, email3).
- As tabelas JOIN (N..N) são transformadas em relacionamentos e as colunas dessas tabelas se tornam propriedades de relacionamento.

Depois de tomar essas medidas para simplificar nosso modelo de banco de dados relacional, aqui está o modelo do modelo de dados em grafo:

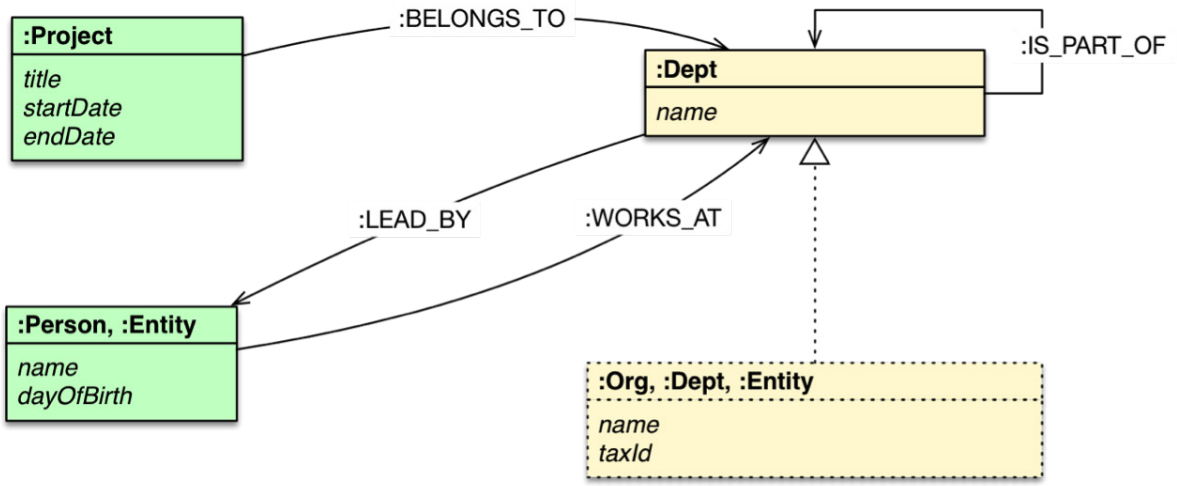

Figura 6: Um modelo de dados orientados a grafos do mesmo domínio com Pessoas e Projetos dentro de uma Organização com vários Departamentos. Com o modelo de grafos, todas as tabelas iniciais JOIN tornaram-se relações de dados.

Este exemplo acima é apenas uma comparação simplificada de um modelo de dados de dados relacionais e grafos. Agora é hora de mergulhar mais profundamente em um exemplo mais extenso tirado de um caso de uso do mundo real.

Estudo de Caso de Modelagem de Dados Relacional vs. Grafos: O Gerenciamento de Data Center

Para mostrar-lhe o verdadeiro poder da modelagem de dados grafos, vamos ver como modelamos um domínio usando técnicas baseadas em relacionamentos e grafos. Você provavelmente já está familiarizado com as técnicas de modelagem de dados RDBMS, então essa comparação irá destacar algumas semelhanças - e muitas diferenças.

Em particular, descobriremos quão fácil é passar de um modelo de grafo conceitual para um modelo de grafo físico e quão pouco o modelo de grafo distorce o que estamos tentando representar em relação ao modelo relacional.

Para facilitar essa comparação, examinaremos um domínio de gerenciamento de data center simples. Neste domínio, vários centros de dados suportam muitas aplicações em nome de muitos clientes que usam diferentes recursos de infra-estrutura, desde máquinas virtuais até equilibradores de carga física.

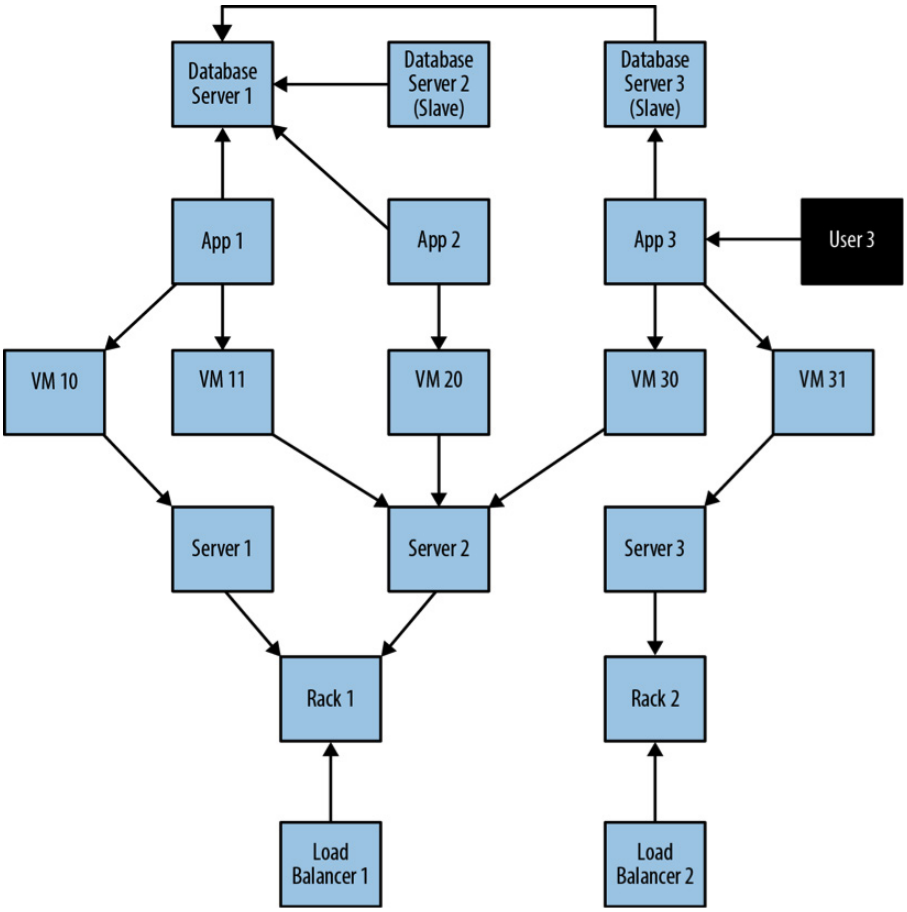

Figura 7: Um pequeno domínio de várias implantações de aplicativos dentro de um data center.

Os bancos de dados orientados à grafos suportam um modelo de dados muito flexível e simples que permite modelar e gerenciar domínios ricos de forma fácil e intuitiva.

Neste exemplo acima, vemos uma visão um tanto simplificada de várias aplicações e a infraestrutura do data center necessária para suportá-las. As aplicações, representadas pelos nós App 1, App 2 e App 3, dependem de um conjunto de bancos de dados denominado Servidor de banco de dados 1, 2, 3.

Embora os usuários dependam logicamente da disponibilidade de uma aplicação e de seus dados, há infraestrutura física adicional entre os usuários e o aplicativo; esta infraestrutura inclui máquinas virtuais (Virtual Machine 10, 11, 20, 30, 31), servidores reais (Servidor 1,2, 3), racks para os servidores (Rack 1, 2) e balanceadores de carga (Load Balancer 1, 2), que suporta as aplicações.

Claro, entre cada um dos componentes há muitos elementos de rede: cabos, switches, painéis de conexão, NICs (controladores de interface de rede), fontes de alimentação, ar condicionado e assim por diante - tudo o que pode falhar em momentos inconvenientes. Para completar a imagem, temos um single user de Straw-man do Application 3, representado pelo User 3.

Como os operadores de um Data Center desses, temos duas preocupações principais:

- Provisão contínua de funcionalidades para atender (ou exceder) um acordo de nível de serviço, incluindo a capacidade de realizar análises prospectivas para determinar pontos de falha únicos e análises retrospectivas para determinar rapidamente a causa de qualquer reclamação de clientes quanto à disponibilidade de serviço.
- Faturamento de recursos consumidos, incluindo o custo de hardware, virtualização, provisionamento de rede e até mesmo os custos de desenvolvimento e operações de software (uma vez que são simplesmente extensões lógicas do sistema que vemos aqui).

Se estamos construindo uma solução de gerenciamento de centro de dados, queremos garantir que o modelo de dados subjacente nos permita armazenar e consultar dados de forma eficiente e direcionada a essas principais preocupações. Também queremos atualizar o modelo subjacente à medida que o portfólio de aplicativos muda, o layout físico do centro de dados evolui e as instâncias da máquina virtual migram.

Dadas essas necessidades e restrições, vejamos como os modelos relacionais e grafos se comparam.

Criando o Modelo Relacional

O primeiro passo na modelagem de dados relacionais é o mesmo que com qualquer outra abordagem de modelagem de dados: entender e concordar sobre as entidades no domínio, como elas inter-relacionam e as regras que regem suas transições de estado.

Esta fase inicial é muitas vezes informal, com muitos esboços de quadro branco e discussões entre especialistas em assuntos e arquitetos de dados. Essas conversas, em seguida, geralmente resultam em diagramas como a Figura 7 acima (que também é um grafo).

O próximo passo é converter este esboço inicial do quadro branco em um diagrama de relacionamento de entidade (E-R) mais rigoroso (que é outro grafo). Transformar o modelo conceitual em um modelo lógico usando uma notação mais estrita nos dá uma segunda chance de refinar nosso vocabulário de domínio para que ele possa ser compartilhado com especialistas em bases de dados relacionais.

[Vale a pena notar que os desenvolvedores adeptos de RDBMS muitas vezes ignoram diretamente o design e a normalização da tabela sem usar um diagrama E-R intermediário].

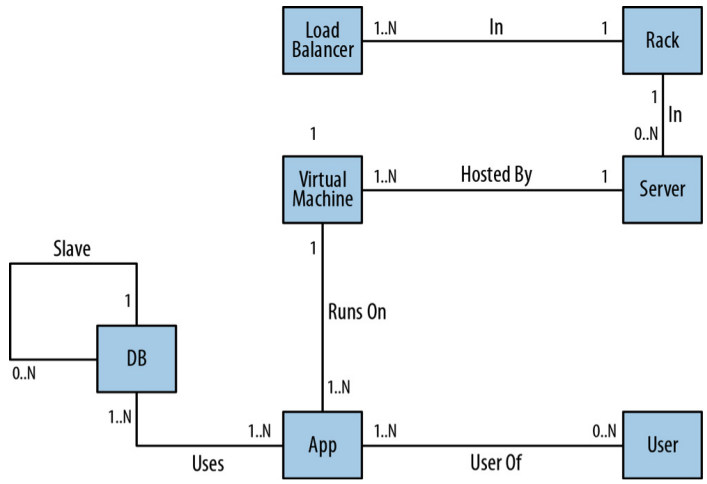

Figura 8: Um diagrama de relação de entidade (E-R) para o nosso domínio de data center.

Agora, com um modelo lógico completo, é hora de mapeá-lo em tabelas e relações, que são normalizadas para eliminar a redundância de dados. Em muitos casos, esta etapa pode ser tão simples como transcrever o diagrama E-R em uma forma tabular e, em seguida, carregar essas tabelas através de comandos SQL no banco de dados.

Mas mesmo o caso mais simples serve para destacar as idiossincrasias do modelo relacional. Por exemplo, na figura abaixo, vemos que uma grande complexidade acidental se introduziu no modelo sob a forma de constraints de chave estrangeira (anunciado como [FK]), que suporta relacionamentos de um para muitos e tabelas de JOIN (por exemplo AppDatabase), que suporta relacionamentos de várias maneiras - e tudo isso antes de adicionar uma única linha de dados de usuários reais.

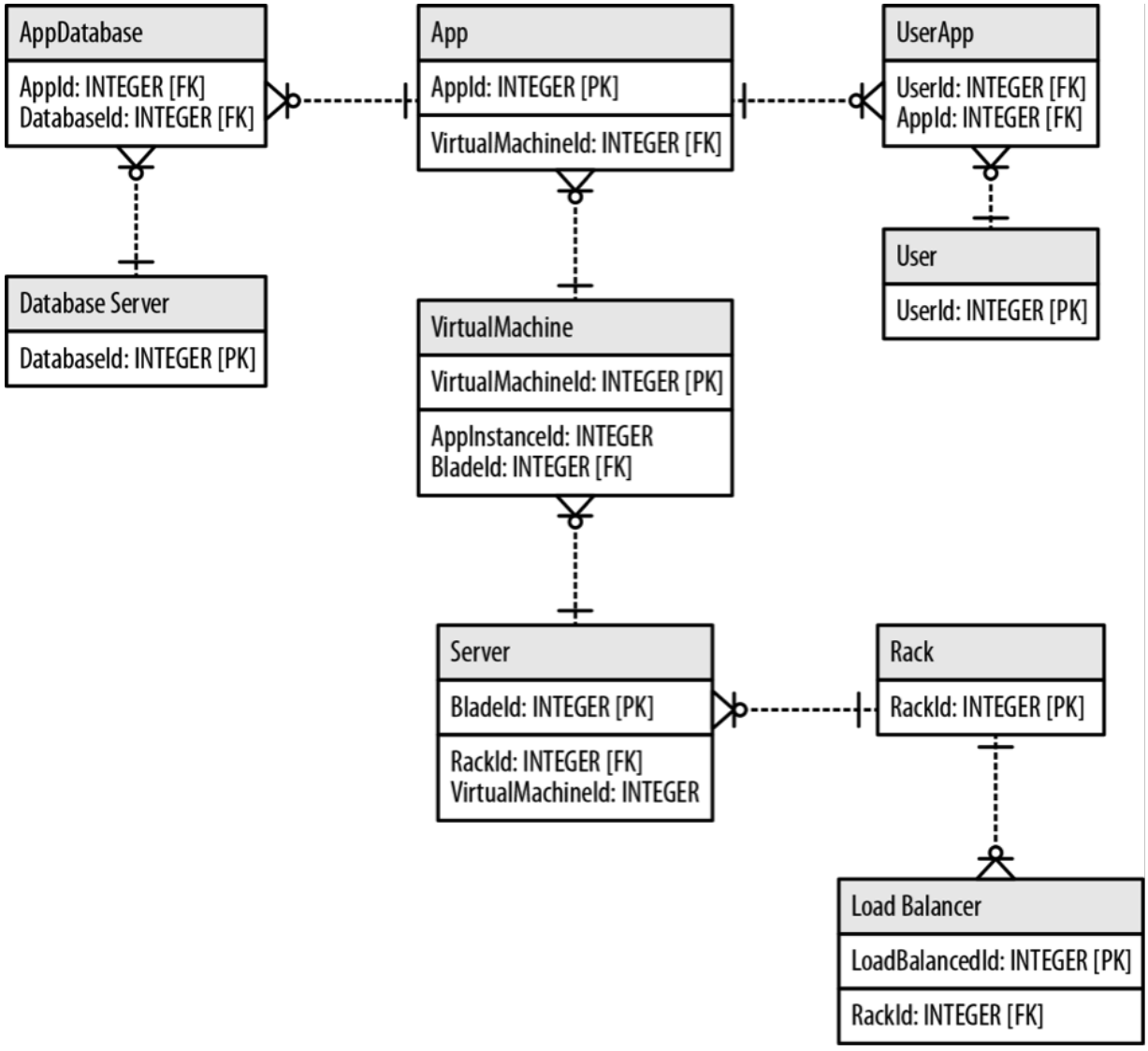

Figura 9: Um modelo de dados relacionais de pleno direito para o nosso domínio do data center.

Essas restrições e complexidades são metadados no nível do modelo que existem simplesmente para que possamos especificar as relações entre as tabelas no tempo de consulta. No entanto, a presença desses dados estruturais é vivamente sentida, porque ele destrói e obscurece os dados do domínio com dados que servem o banco de dados, e não o usuário.

O Problema da Desnormalização do Modelo de Dados Relacionais

Até agora, agora temos um modelo de dados relacional normalizado que é relativamente fiel ao domínio, mas nosso trabalho de design ainda não está completo.

Um dos desafios do paradigma relacional é que os modelos normalizados geralmente não são suficientemente rápidos para as necessidades do mundo real. Em teoria, um esquema normalizado é adequado para responder qualquer tipo de consulta ad hoc que representamos para o domínio, mas, na prática, o modelo deve ser adaptado para padrões de acesso específicos.

Em outras palavras, para que os bancos de dados relacionais funcionem bem o suficiente para necessidades regulares de aplicativos, devemos abandonar qualquer vestígio de afinidade de domínio verdadeiro e aceitar que devemos alterar o modelo de dados do usuário para se adequar ao mecanismo de banco de dados e não ao usuário. Essa abordagem é chamada de desnormalização.

A desnormalização envolve a duplicação de dados (substancialmente em alguns casos) para obter o melhor desempenho da consulta.

Por exemplo, considere um lote de usuários e seus detalhes de contato. Um usuário típico frequentemente tem vários endereços de e-mail, que normalmente armazenamos em uma tabela EMAIL separada. No entanto, para reduzir a penalidade de desempenho de realizar JOIN entre duas tabelas, é bastante comum adicionar uma ou mais colunas dentro da tabela USER para armazenar os endereços de e-mail mais importantes de um usuário, como Email1, Email2, Email3 etc.

Assumindo que cada desenvolvedor no projeto entende o modelo de dados desnormalizado e como ele mapeia seu código centrado no domínio (o que é uma grande hipótese), a desnormalização não é uma tarefa trivial.

Muitas vezes, as equipes de desenvolvimento se voltam para um especialista RDBMS para transformar um modelo normalizado em um desnormalizado que alinha com as características do RDBMS subjacente e do nível de armazenamento físico. Fazer tudo isso envolve uma quantidade substancial de redundância de dados.

O Custo da Rápida Mudança no Modelo Relacional

É fácil pensar que o processo design-normalização-desnormalização é aceitável porque é apenas uma tarefa única. Afinal, o custo desse trabalho inicial vale a pena na vida útil do sistema, certo? Errado.

Embora esta ideia inicial única seja atraente, não combina a realidade do processo de desenvolvimento ágil atual. Os sistemas mudam frequentemente - não só durante o desenvolvimento, mas também durante a vida útil da produção.

Embora a maioria dos sistemas gaste a maior parte do tempo em ambientes de produção, esses ambientes raramente são sempre estáveis. A mudança de requisitos de negócios e os requisitos regulatórios evoluem, então nossos modelos de dados também devem.

Adaptar o nosso modelo de banco de dados relacional requer uma mudança estrutural conhecida como migração. As migrações fornecem uma abordagem estruturada e passo a passo para as refatorações de banco de dados para que ele possa evoluir para atender aos requisitos em mudança. Ao contrário das refatorações de código - que tipicamente levam uma questão de minutos ou segundos - as refatorações de banco de dados podem levar semanas ou meses para serem concluídas, com o tempo de inatividade para mudanças de esquema.

O principal problema com o modelo relacional desnormalizado é a sua resistência à evolução rápida que o negócio de hoje causa às aplicações. Como vimos neste exemplo do centro de dados, as mudanças impostas ao modelo inicial do quadro branco do início ao fim criam um fosso crescente entre o mundo conceitual e a forma como os dados são dispostos fisicamente.

Essa dissonância conceitual-relacional evita que empresas e outras partes interessadas não-técnicas colaborem mais sobre a evolução do sistema. Como resultado, a evolução da aplicação está significativamente atrasada na evolução do negócio.

Agora que examinamos minuciosamente o processo de modelagem de dados relacionais, vamos abordar a abordagem de modelagem de dados grafos.

Criando o Modelo de Dados Orientado à Grafos

Como vimos, a modelagem de dados relacionais divide o modelo de armazenamento de uma aplicação da visão conceitual de seus stakeholders.

Os bancos de dados relacionais - com seus esquemas rígidos e características de modelagem complexas - não são uma ferramenta especialmente boa para suportar mudanças rápidas. O que precisamos é de um modelo que está alinhado com o domínio, mas que não sacrifique o desempenho e que ajude a evolução, mantendo a integridade dos dados, pois sofre mudanças e crescimento rápidos.

Esse modelo é o modelo orientado à grafos. Como, então, o processo de modelagem de dados é diferente? Vamos começar.

Nos estágios iniciais da modelagem de grafos, o trabalho é semelhante à abordagem relacional: usando métodos simples como esboços de quadro branco, descrevemos e concordamos com o domínio inicial. Depois disso, as metodologias de modelagem de dados divergem.

Mais uma vez, aqui está o nosso exemplo de domínio do centro de dados modelado no quadro branco:

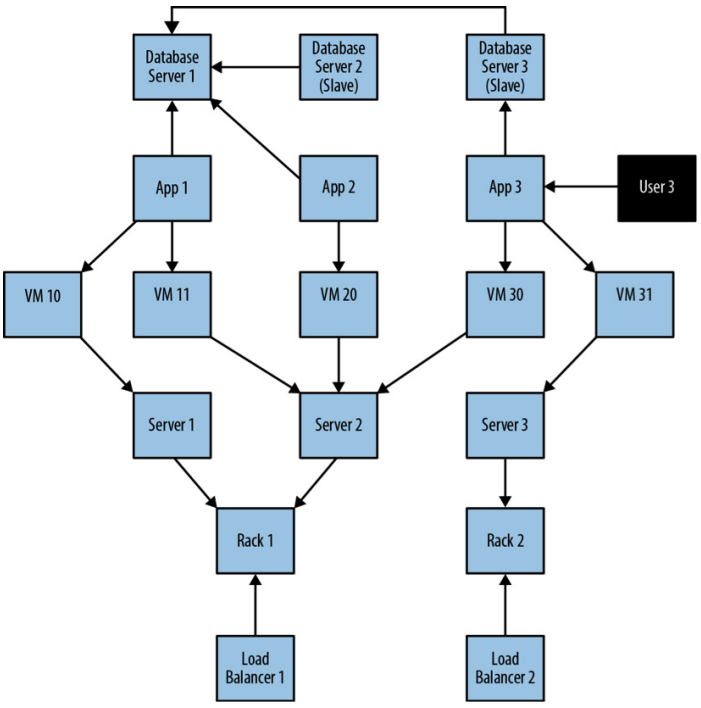

Figura 10: Nosso exemplo de domínio do centro de dados com várias implantações de aplicativos.

Em vez de transformar nosso modelo em tabelas, nosso próximo passo é simplesmente enriquecer nossa estrutura já pensada em grafos. Este enriquecimento visa representar de forma mais precisa nossos objetivos de aplicação no modelo de dados. Em particular, vamos capturar papéis relevantes como labels, atributos como propriedades e conexões para entidades vizinhas como relacionamentos.

Ao enriquecer este modelo de domínio de primeira rodada com propriedades e relacionamentos adicionais, produzimos um modelo de grafo em sintonia com nossas necessidades de dados; ou seja, nós construímos nosso modelo para responder aos tipos de perguntas que nossa aplicação irá solicitar aos seus dados.

Para esclarecer o nosso modelo de dados orientado à grafos em desenvolvimento, precisamos garantir um contexto semântico correto. Fazemos isso criando relacionamentos nomeados e direcionados entre nós para capturar os aspectos estruturais do nosso domínio.

Logicamente, é tudo o que precisamos fazer. Sem tabelas, sem normalização, sem desnormalização. Uma vez que temos uma representação precisa do nosso modelo de domínio, movê-lo para o banco de dados é trivial.

O Pulo do Gato

Então, qual é a grande idéia? Com um banco de dados orientado à grafos, o que você esboça no quadro branco é o que você armazena no banco de dados. É simples assim.

Depois de adicionar propriedades, Labels e relacionamentos, o modelo de grafo resultante para o cenário do centro de dados parece assim:

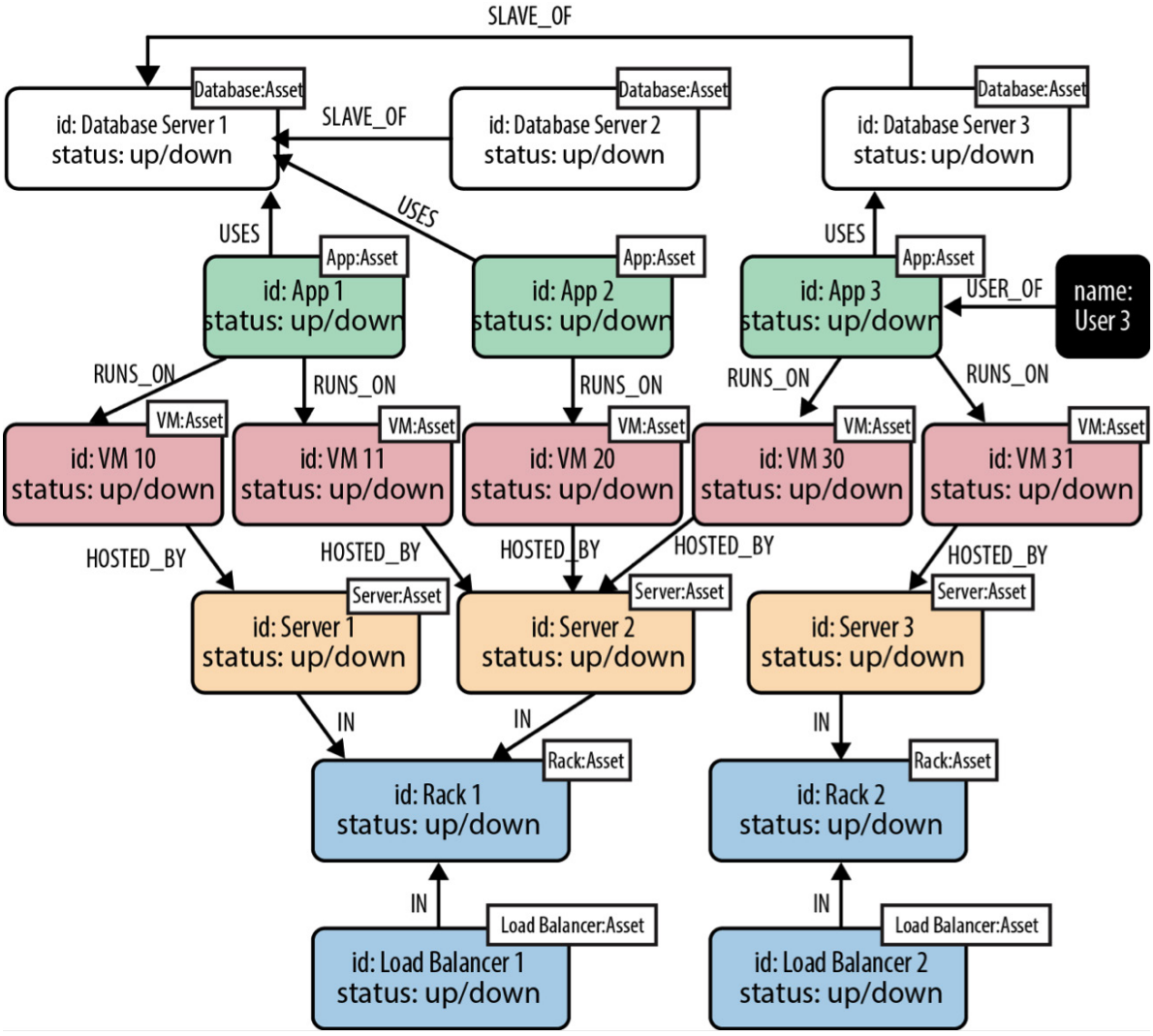

Figura 11: Um modelo de dados orientado à grafos completo para o nosso domínio do centro de dados.

Observe que a maioria dos nós aqui tem dois Labels: tanto um Label de tipo específico (como Base de dados, Aplicação ou Servidor), e um Label de Ativo de uso geral. Isso nos permite segmentar tipos específicos de ativos com algumas de nossas consultas e todos os ativos - independentemente do tipo - com outras consultas.

Comparado com o modelo de banco de dados relacional concluído (incluído novamente logo abaixo), pergunte-se qual dos dois modelos de dados é mais fácil de evoluir, contém relacionamentos mais ricos e, no entanto, ainda é simples o suficiente para os interessados das empresas entenderem. Nós pensamos assim:

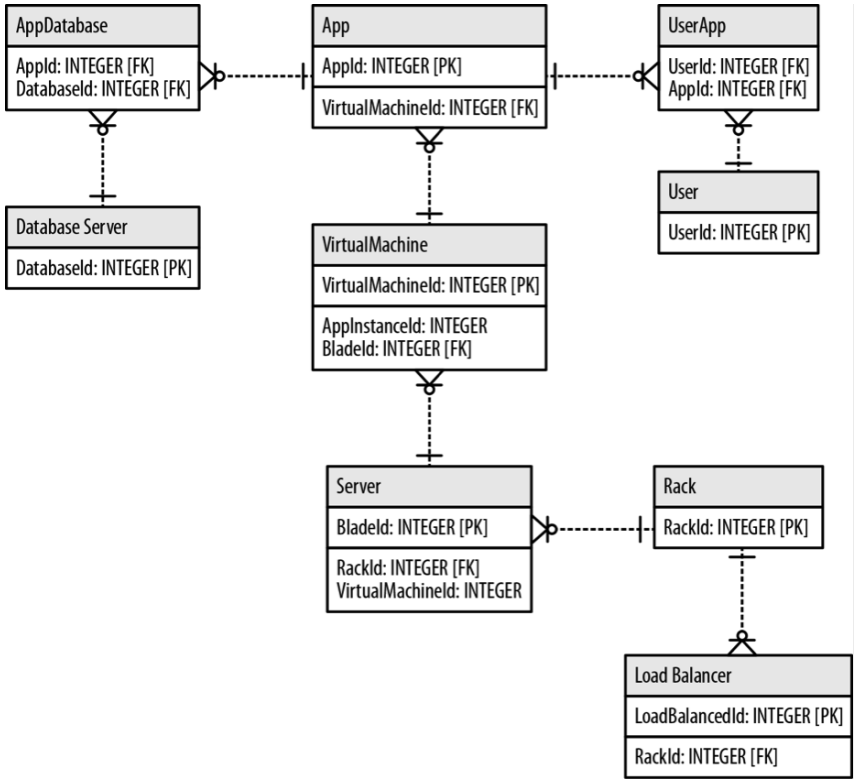

Figura 12: Um modelo de dados relacionais de pleno direito para o nosso domínio do data center. Este modelo de dados é significativamente mais complexo - e menos amigável - do que o nosso modelo de grafo na página anterior.

Conclusão

Não esqueça: modelos de dados estão sempre mudando. Embora possa ser fácil descartar uma dor de cabeça de modelo antecipada com a ideia de que você nunca mais terá de novo, as práticas de desenvolvimento ágeis de hoje terão você de volta no quadro branco (ou pior, chamando um especialista em migração) mais cedo do que você pensa.

Mesmo assim, a modelagem de dados é apenas parte do desenvolvimento do ciclo de vida do banco de dados. Ser capaz de consultar seus dados com facilidade e eficiência - o que muitas vezes significa tempo real - é tão importante como ter um modelo de dados rico e flexível. No próximo capítulo, examinaremos as diferenças entre linguagens de consulta para RDBMS e linguagens de consulta de banco de dados orientados a grafo.

Capítulo 4:
Query Languages: SQL vs. Cypher

A eficiência das consultas baseadas em grafos se dá ao fato que elas são executadas em tempo real e em uma economia que funciona na velocidade de um único tweet, essa é uma diferença fundamental que você não pode ignorar.

Quando se trata de uma linguagem de consulta de banco de dados, a eficiência linguística é importante.

Consultar bancos de dados relacionais é fácil com o SQL. Como linguagem de consulta declarativa, o SQL permite tanto a consulta ad hoc fácil em uma ferramenta de banco de dados quanto a especificação de consultas relacionadas a casos de uso em seu código. Mesmo os mapeadores objeto-relacionais (ORM) utilizam a linguagem SQL sob o capô para conversar com o banco de dados.

Mas o SQL corre contra os principais desafios de desempenho quando tenta navegar dados conectados. Para questões de relacionamento com dados, uma única consulta no SQL pode ser muitas linhas a mais do que a mesma consulta em uma linguagem de consulta de banco de dados como Cypher (mais sobre Cypher abaixo).

As longas consultas SQL não só levam mais tempo para serem executadas, mas também são mais propensas a incluir erros de codificação humana por sua complexidade. Além disso, consultas mais curtas aumentam a facilidade de compreensão e manutenção em sua equipe de desenvolvedores. Por exemplo, imagine se um desenvolvedor externo teve que lidar uma consulta SQL complicada e tentar descobrir a intenção do desenvolvedor original – o tamanho do problema em que isso resultaria.

Mas, de que nível de ganhos de eficiência falamos entre consultas SQL e consultas de grafo? Quanto mais eficiente é um contra o outro? A resposta: rápido o suficiente para fazer uma diferença significativa para sua organização.

O Relacionamento Crítico Entre Idiomas de Consulta e Modelos de Dados

Vale ressaltar que um idioma de consulta não é apenas perguntar ao banco de dados para um determinado conjunto de resultados; também é sobre a modelagem desses dados, em primeiro lugar.

Sabemos do capítulo anterior que a modelagem de dados para uma base de dados orientado à grafos é tão fácil como conectar círculos e linhas em um quadro branco. O que você esboça no quadro branco é o que você armazena no banco de dados.

Por si só, esta facilidade de modelagem tem muitos benefícios empresariais, o mais óbvio é que você pode entender o que seus desenvolvedores de banco de dados estão realmente criando. Mas há mais: um modelo intuitivo criado com a linguagem de consulta correta garante que não há incompatibilidade entre a forma como você construiu o modelo de dados e como você o analisa.

Uma linguagem de consulta representa seu modelo de perto. É por isso que o SQL trabalha focado sobre tabelas e JOINs, enquanto o Cypher trabalha sobre relações entre entidades. Tanto quanto o modelo orientado à grafos é mais natural para trabalhar, o Cypher também toma emprestado a partir da representação pictórica de círculos relacionados com setas que qualquer interessado (seja técnico ou não técnico) possa entender.

Em um banco de dados relacional, o processo de modelagem de dados é, até agora, abstraído das consultas SQL atuais do dia a dia, que existe uma grande disparidade entre análise e implementação. Em outras palavras, o processo de construção de um modelo de banco de dados relacional não é adequado para perguntar (e responder) perguntas com eficiência desse mesmo modelo.

Os modelos de banco de dados orientado à grafos, por outro lado, não só comunicam como seus dados estão relacionados, mas também ajudam você a comunicar claramente os tipos de perguntas que deseja solicitar ao seu modelo de dados. Modelo de dados orientado à grafos e consultas orientadas à grafos são apenas dois lados da mesma moeda.

A linguagem de consulta correta do banco de dados nos ajuda a atravessar ambos os lados.

Uma Introdução ao Cypher, a Linguagem de Consulta de Grafos

Assim como o SQL é a linguagem de consulta padrão para bancos de dados relacionais, o Cypher é uma linguagem de consulta para tecnologias que utilizam grafos. O advento do projeto openCypher expandiu o alcance da Cypher bem além do Neo4j, seu patrocinador original.

O Cypher - também uma linguagem de consulta declarativa - é construído sobre os conceitos básicos e as cláusulas do SQL, mas com função adicional específica do grafo, tornando simples trabalhar com um modelo de grafo rico sem ser excessivamente detalhado.

(Nota: Esta introdução não é um documento de referência para Cypher, mas apenas uma visão geral de alto nível).

O Cypher é projetado para ser facilmente lido e compreendido por desenvolvedores, profissionais de banco de dados e partes interessadas de negócios. É fácil de usar, pois corresponde a maneira como descrevemos os grafos usando diagramas intuitivamente.

Se você já tentou escrever uma instrução SQL com um grande número de JOINs, você sabe que você rapidamente perde de vista o que a consulta realmente faz, devido a todo o ruído técnico. Em contraste, a sintaxe Cypher permanece limpa e focada em conceitos de domínio, uma vez que as consultas são expressas visualmente.

A noção básica de Cypher é que ele permite que você pergunte ao banco de dados para encontrar dados que correspondam a um padrão específico. De forma coloquial, podemos pedir ao banco de dados que “encontre coisas assim”, e a maneira como descrevemos o aspecto de “coisas assim” é desenhá-las usando a arte ASCII.

Considere o grafo social abaixo descrevendo três amigos em comum:

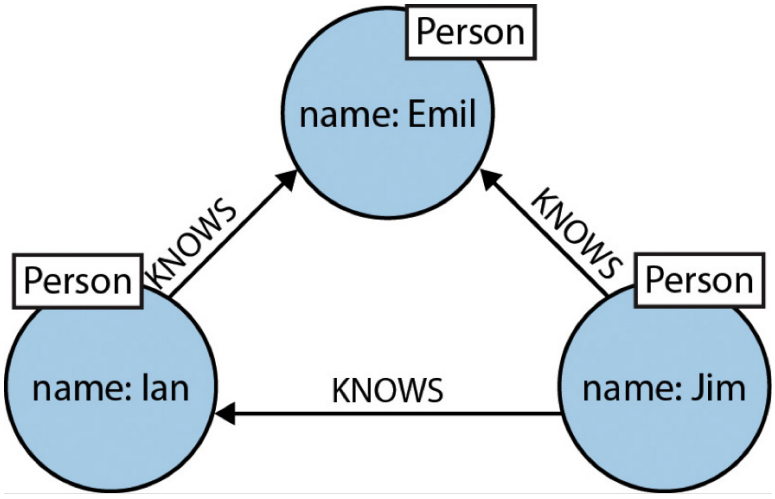

Figura 13: Um grafo social que descreve a relação entre três amigos.

Se quisermos expressar o padrão deste grafo básico em Cypher, escreveríamos:

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

Esta declaração Cypher descreve um caminho que forma um triângulo que conecta um nó que chamamos Jim para os dois nós que chamamos Ian e Emil, e que também conecta o nó Ian ao nó Emil. Como você pode ver, Cypher segue naturalmente a maneira como desenhamos grafos no quadro branco.

Agora, enquanto este padrão Cypher descreve uma estrutura de grafo simples, ele ainda não se refere a nenhum dado específico em um banco de dados orientados a grafo . Para vincular o padrão a nós e relacionamentos específicos em um conjunto de dados existente, primeiro precisamos especificar alguns valores de propriedade e Labels de nós que ajudem a localizar os elementos relevantes no conjunto de dados:

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

Aqui está a declaração mais atual de Cypher:

Aqui encadeamos cada nó ao seu identificador usando sua propriedade de nome e Label Pessoa. O identificador de Emil, por exemplo, é vinculado a um nó no conjunto de dados com um Label Pessoa e uma propriedade de nome cujo valor é Emil. A ancoragem de partes do padrão a dados reais dessa maneira é prática normal do Cypher.

O Guia do Desenvolvedor RDBMS para Cláusulas Cypher

Como a maioria das linguagens de consulta, o Cypher é composto de cláusulas.

As consultas mais simples consistem em uma cláusula MATCH seguida por uma cláusula RETURN. Aqui está um exemplo de uma consulta Cypher que usa essas duas cláusulas para encontrar os amigos comuns de um usuário chamado Jim (do nosso grafo social retratado na página anterior):

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

Vejamos cada uma das cláusulas com mais detalhes:

MATCH

A cláusula MATCH está no cerne da maioria das consultas Cypher.

Usando caracteres ASCII para representar nós e relacionamentos, desenhamos os dados que nos interessam. Nós desenhamos nós com parênteses, como no exemplo da consulta acima:

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

Criamos relacionamentos usando pares de traços com sinais maiores (ou menos) que os sinais (-> e <-) onde os sinais < e > indicam a direção do relacionamento. Entre os traços, os nomes das relações são entre parêntesis retos (colchetes) e prefixados por dois pontos, como neste exemplo a partir da consulta acima:

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

As Labels de nó também são prefixadas por dois pontos. Como você vê no primeiro nó da consulta, Pessoa é a Label aplicável:

```
(a:Person … )
```

Os pares de valores-chave de propriedade do nó (e relacionamento) são então especificados em chaves curly, como neste exemplo:

```
( … {name:'Jim'})
```

Na nossa consulta de exemplo original, estamos procurando por um nó denominado Pessoa com uma propriedade de nome cujo valor seja Jim. O valor de retorno dessa pesquisa é vinculado ao identificador a. Este identificador nos permite consultar o nó que representa Jim durante o resto da consulta.

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

Vale ressaltar que esse padrão poderia, em teoria, ocorrer muitas vezes em todo o nosso grafo, especialmente em um grande conjunto de dados de usuários.

Para limitar a consulta, precisamos ancorar alguma parte dela em um ou mais lugares no grafo. Ao especificar que estamos procurando por um nó denominado Pessoa cujo valor de propriedade de nome seja Jim, vinculamos o padrão a um nó específico no grafo - o nó representando Jim.

Cypher corresponde ao restante do padrão ao grafo que circunda imediatamente esse ponto de ancoragem com base nas informações fornecidas sobre relacionamentos e nós vizinhos. Ao fazê-lo, descobre os nós para se ligar aos outros identificadores. Enquanto um sempre estará ancorado em Jim, b e c estarão vinculados a uma sequência de nós à medida que a consulta for executada.

RETURN

Esta cláusula especifica quais expressões, relacionamentos e propriedades nos dados correspondentes devem ser devolvidos ao cliente. Na nossa consulta de exemplo, estamos interessados em retornar os nós vinculados aos identificadores b e c.

Outras Cláusulas Cypher

Outras cláusulas que você pode usar em uma consulta Cypher incluem:

WHERE
Fornece critérios para filtrar resultados de correspondência de padrões.

CREATE e CREATE UNIQUE
Cria nós e relacionamentos.

MERGE
Garante que o padrão fornecido existe no grafo, seja reutilizando nós existentes e relacionamentos que correspondem aos predicados fornecidos, ou criando novos nós e relacionamentos. Seria como um CREATE, porém só criaria se não existisse algo anteriormente.

DELETE/REMOVE
Remove nós, relacionamentos e propriedades.

SET
Define os valores e as labels das propriedades.

ORDER BY
Classifica resultados como parte de um RETURN.

SKIP LIMIT
Salta os resultados que seriam resultados e limita o número de resultados retornados.

FOREACH
Executa uma ação de atualização para cada elemento em uma lista.

UNION
Combina resultados de duas ou mais consultas.

WITH
Cadeias subsequentes consulta peças e encaminha resultados de um para o outro. Semelhante aos comandos de tubulação no Unix.

Se essas cláusulas parecerem familiares - especialmente se você é desenvolvedor de RDBMS - isso é ótimo! Cypher destina-se a ser fácil de aprender para os veteranos do SQL, ao mesmo tempo que é simples o suficiente para iniciantes. (Clique aqui para obter o cartão de visita Cypher mais atualizado para mergulhar mais profundamente na linguagem de consulta Cypher.)

SQL vs. Cypher Query : O bom, o Mau e o Feio

Agora que você tem uma compreensão básica do Cypher, é hora de compará-lo lado a lado com o SQL para perceber a eficiência linguística do primeiro - e a ineficiência deste último - quando se trata de consultas em torno de dados conectados.

Nosso primeiro exemplo usa o domínio organizacional (do Capítulo 3) ilustrado abaixo como um modelo de dados relacional:

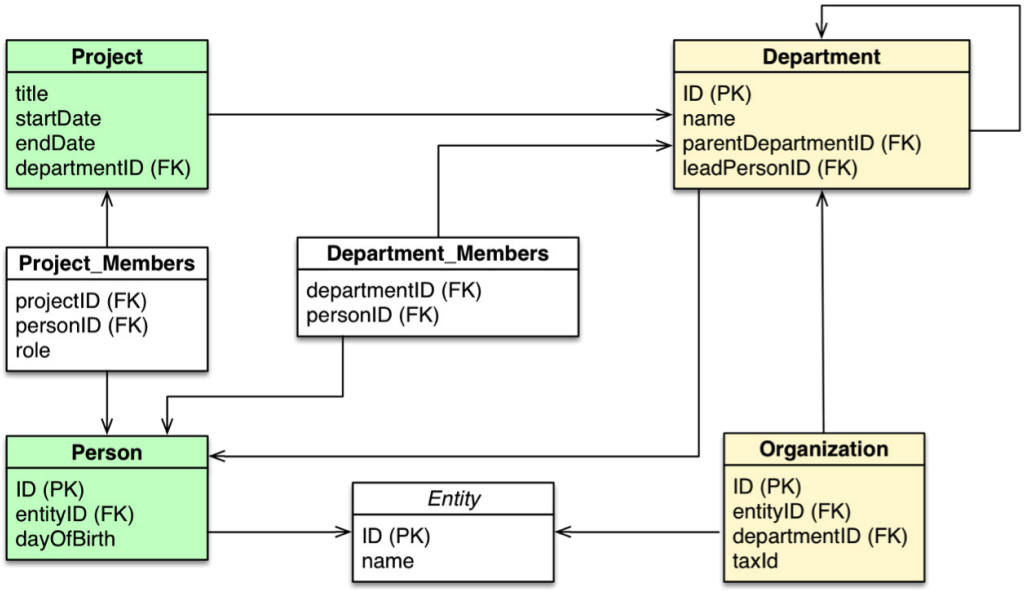

Figura 14: Um modelo de dados relacionais de um domínio organizacional.

No domínio organizacional representado no modelo acima, o que seria uma declaração SQL que alista os funcionários no "Departamento de TI"? E como essa afirmação se compara a uma declaração Cypher?

Declaração SQL:

```
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"
```

Declaração Cypher:

```
MATCH (p:Person)<-[:EMPLOYEE]-(d:Department)
WHERE d.name = "IT Department"
RETURN p.name
```

Neste exemplo na página anterior, a consulta Cypher é metade do comprimento da instrução SQL e significativamente mais simples. Não só essa consulta Cypher seria mais rápida para criar e executar, mas também reduz as chances de erro.

Agora, para outro exemplo, esse mais extremo. Começaremos com a consulta do Cypher:

```
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;
```

Esta consulta Cypher diz que para cada cliente que comprou um produto, veja os produtos que os clientes similares compraram e adicione-os como recomendações. A cláusula WHERE remove produtos que o cliente já comprou, uma vez que não queremos recomendar algo que o cliente já comprou.

Cada uma das setas na cláusula MATCH da consulta Cypher representa um relacionamento que seria modelado como uma tabela JOIN de muitos para muitos em um modelo relacional com dois JOINs cada. Portanto, mesmo essa consulta simples abrange potencialmente seis JOINs em tabelas. Aqui está a consulta equivalente no SQL:

```
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
```

Esta instrução SQL é três vezes maior que a consulta Cypher equivalente. Não só sofrerá problemas de desempenho devido à complexidade JOIN, mas também se degradará no desempenho à medida que o conjunto de dados crescer.

Conclusão

Se o desempenho do aplicativo for uma prioridade, o idioma da consulta do banco de dados é importante.

O SQL é bem otimizado para modelos de banco de dados relacionais, mas uma vez que ele tem que lidar com consultas complexas e orientadas para relacionamento, seu desempenho se degrada rapidamente. Nesses casos, o problema de raiz não está no SQL, mas com o próprio modelo relacional, que não foi projetado para lidar com dados conectados.

Para domínios com dados altamente conectados, o modelo de grafo é imprescindível e, como resultado, é também uma linguagem de consulta de grafos como Cypher. Se o seu time de desenvolvimento vem de um fundo SQL, o Cypher será fácil de aprender e ainda mais fácil de executar.

No que diz respeito ao seu próximo aplicativo de nível corporativo, você ficará feliz por o fato de a linguagem de consulta estar embutida ser construída para velocidade e eficiência.

Capítulo 5:
Importando Dados: do RDBMS Para o Grafo

Se você está pronto para mover seu RDBMS de legado completo em um banco de dados orientado à grafos, você está sincronizando bancos de dados para a persistência de políglota ou apenas está realizando uma breve prova de conceito, em algum momento você deseja mover dados de seu banco de dados relacional em um banco de dados orientados a grafo .

Neste capítulo, mostraremos como fazer esse processo tão suave e transparente quanto possível.

Seu primeiro passo é garantir que você tenha uma compreensão adequada do modelo de dados orientados a grafos (ou seja, nós, relacionamentos, Labels, propriedades e tipos de relacionamento), particularmente quando se aplica ao seu domínio dado.

Na verdade, você deve pelo menos completar um modelo de grafo básico em um quadro branco antes de começar a importação de dados. Conhecer o seu modelo de dados antes do tempo torna o processo de importação significativamente menos doloroso.

Abordagens estratégicas para a importação de dados RDBMS

Existem três abordagens principais para mover dados para um banco de dados orientados a grafo de um relacional. Qual abordagem é melhor para sua aplicação ou arquitetura depende de seus objetivos particulares.

Abaixo, você pode ver cada uma das abordagens para lidar com dados entre um banco de dados relacional e grafo.

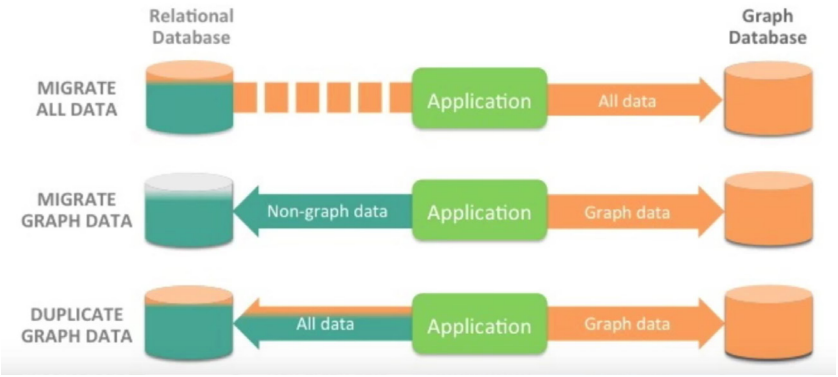

Figura 15: As três abordagens mais comuns para o tratamento de dados entre um banco de dados relacional e um grafo.

Em primeiro lugar, algumas equipes de desenvolvimento migram todos os seus dados do seu banco de dados relacional antigo para um banco de dados orientados a grafo. Isso geralmente é uma migração em massa única.

Em segundo lugar, outros desenvolvedores continuam a usar seu banco de dados relacional para qualquer caso de uso que dependa de dados suplementares. Então, para quaisquer casos de uso que envolvam muitos JOINs ou relacionamentos de dados, eles armazenam esses dados em um banco de dados orientados a grafo.

Em terceiro lugar, algumas equipes de desenvolvimento persistem todos os seus dados

em um banco de dados relacional e em um banco de dados orientados a grafo. Dessa forma, os dados podem ser consultados em qualquer forma que seja o mais ideal para as consultas que estão tentando executar.

Nenhuma destas é a abordagem "correta" para gerenciar dados entre um RDBMS e um grafo. Sua equipe deve considerar os objetivos de sua aplicação, casos de uso freqüente e consultas mais comuns e escolher a solução apropriada para seu ambiente particular.

Extraindo Seus Dados de um RDBMS

Se você decidir que precisa importar seus dados relacionais para um banco de dados orientados a grafo, o primeiro passo é extraí-lo do seu RDBMS existente.

A maioria dos bancos de dados relacionais permitem que você despeje as mesas inteiras ou conjuntos de dados inteiros, além de exportar CSV e postar consultas. Essas tarefas geralmente são apenas uma função de cópia do próprio banco de dados. Claro, em muitos casos, o arquivo ACC reside no banco de dados, então você deve fazer o download a partir daí, o que pode ser um desafio. Outra opção é acessar seu banco de dados relacional com um driver de banco de dados como JDBC ou outro driver para extrair os conjuntos de dados que deseja retirar.

Além disso, se você quiser configurar um mecanismo de sincronização entre seus bancos de dados relacionais e orientados a grafos, então faz sentido puxar regularmente os dados dados de acordo com um timestamp / hora ou outro sinalizador atualizado para que os dados sejam sincronizados em seu grafo.

Outra faceta a considerar é que muitos bancos de dados relacionais não foram projetados ou otimizados para exportar grandes quantidades de dados dentro de um curto período de tempo. Então, se você estiver tentando migrar dados diretamente de um RDBMS para um grafo, o processo pode paralisar significativamente.Por exemplo, em um caso, um cliente Neo4j tinha uma grande rede social armazenada em um cluster MySQL. Exportar os dados do banco de dados MySQL levou três dias; importá-lo para Neo4j levou apenas três horas.

Uma dica final antes de começar sua importação: quando você escreve no disco, certifique-se de desativar os scanners de vírus e verificar a programação do seu disco para que você obtenha o maior desempenho de disco possível. Também vale a pena verificar ou definir outras opções que possam aumentar o desempenho durante o processo de importação.

Importando dados via LOAD CSV

A maneira mais fácil de importar dados de seu banco de dados relacional é criar um dump CSV de tabelas de entidades individuais e tabelas de JOIN. O formato CSV é o menor denominador comum de formatos de dados entre uma variedade de aplicações diferentes. Embora o próprio formato CSV seja impopular, também é o mais fácil de trabalhar quando se trata de importar dados em um banco de dados orientados a grafo.

Em Neo4j especificamente, LOAD CSV é uma palavra-chave Cypher que permite carregar arquivos CSV de HTTP ou URLs de arquivos em seu banco de dados. Cada linha de dados está disponível para sua declaração de Cypher e, a partir dessas linhas, você pode realmente criar ou atualizar nós e relacionamentos dentro do seu grafo.

O comando LOAD CSV é uma maneira poderosa de converter dados planos (ou seja, arquivos CSV) em dados de grafo conectados. LOAD CSV funciona tanto com arquivos CSV de tabela única quanto com arquivos que contêm uma tabela totalmente desnormalizada ou um JOIN de várias tabelas.

LOAD CSV permite converter, filtrar ou desestruturar dados de importação durante o processo de importação. Você também pode usar esse comando para dividir áreas, puxar um único valor ou iterar sobre uma certa lista de atributos e, em seguida, filtrá-los como atributos.

Finalmente, com o LOAD CSV, você pode controlar o tamanho das transações para que você não se depare com problemas de memória com uma determinada palavra-chave e você pode executar o LOAD CSV através do shell Neo4j (e não apenas o navegador Neo4j), o que torna mais fácil escrever os scripts das suas importações de dados.

Em resumo, você pode usar o comando LOAD CSV do Cypher para:

- Ingerir dados, acessar colunas por nome de cabeçalho ou offset
- Converter valores de strings para diferentes formatos e estruturas (toFloat, split, ...)
- Saltar as linhas a serem ignoradas
- MATCH nós existentes com base em pesquisas de atributos
- CREATE ou MERGE nós e relacionamentos com Labels e atributos dos dados da linha

- CONFIGURAR novAs Labels e propriedades ou REMOVER os desatualizados

Um exemplo de LOAD CSV

Aqui está um breve exemplo de importação de um arquivo CSV no Neo4j usando o comando LOAD CSV Cypher chamado person.csv e sua estrutura de comando em cypher respectivamente:

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

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

Você pode importar vários arquivos CSV de uma ou mais fontes de dados (incluindo o seu RDBMS) para enriquecer seu modelo de domínio principal com outras informações que possam adicionar informações e recursos interessantes.

Outras ferramentas de importação dedicadas ajudam você a importar volumes maiores (10M + linhas) de dados de forma eficiente, conforme descrito abaixo.

O carregador de massa de linha de comando

O comando neo4j-import é uma ferramenta de entrada escalável para inserções em massa. Esta ferramenta leva os arquivos CSV e os escalas em todas as suas CPUs e capacidade de disco disponíveis, colocando os dados em uma arquitetura de estágios onde cada etapa de entrada é paralelizada, se possível.

Em seguida, a ferramenta mostra a entrada passo a passo usando alguma compressão avançada na memória para criar novas estruturas gráficas.

O carregador de massa de linha de comando é rápido, capaz de importar até um milhão de registros por segundo e lidar com grandes conjuntos de dados de vários bilhões de nós, relacionamentos e propriedades. O comando neo4j-import é particularmente útil para a população inicial do banco de dados.

O Carregador Personalizado Baseado em Cypher

Em um teste, um milhão de nós e relações por segundo foram inseridos com declarações de Cypher altamente concorrentes usando este método.

Para importar dados, você também pode usar as APIs Neo4j para executar declarações do Cypher você mesmo. Com essas APIs, você pode executar, criar, atualizar e mesclar

instruções usando o Cypher.

Uma API é o endpoint HTTP, que está disponível para todos os drivers. Você também pode usar o ponto final HTTP diretamente de um cliente HTTP ou uma biblioteca HTTP em seu idioma.

Usando o endpoint HTTP (ou outra API), você pode extrair os dados de seu banco de dados relacional (ou outra fonte de dados) e convertê-lo em parâmetros para instruções Cypher. Então você pode lidar e controlar as transações de importação a partir daí.

Do Neo4j 2.2 em diante, o carregador personalizado baseado em Cypher também funciona muito bem com gravações altamente concorrentes.

O método de carregamento baseado em Cypher funciona com vários drivers diferentes, incluindo o driver JDBC. Se você tem uma ferramenta ETL ou um programa Java que já usa uma ferramenta JDBC, você pode usar o driver JDBC do Neo4j para importar dados no Neo4j porque as instruções do Cypher são apenas cadeias de consulta. Nesse cenário, você também pode fornecer parâmetros para suas afirmações Cypher.

Outros recursos de importação RDBMS-to-Graph

Este capítulo apenas cobriu os três métodos mais comuns para importar dados em um banco de dados orientados a grafo de uma loja relacional. Os seguintes são recursos adicionais sobre métodos adicionais para importação de dados, além de guias mais detalhados sobre os três métodos discutidos acima:

- Manual: LOAD CSV
- Guia: importação de dados
- Guia: Importação CSV
- Ferramenta: importação RDBMS direta
- Ferramenta: importação de SQL para Neo4j
- Blog: Importando dados do AdventureWorks para o Neo4j
- Neo4j para MetaData Relacional (SQLServer)

Este ebook apenas arranha a superfície quando se trata de como o desenvolvedor RDBMS de hoje pode aproveitar melhor os bancos de dados orientados à grafos.

Recursos Adicionais:
Mais Sobre Banco de Dados Orientados a Grafos Para o Desenvolvedor RDBMS

Para obter mais informações, dicas e truques na interseção de bancos de dados relacionais e orientados a grafos, explore qualquer um dos muitos recursos abaixo:

- A série de blogs Graph Databases for Beginners
- Guia do desenvolvedor: do relacionamento ao Neo4j
- O livro de eReilly's Graph Databases ebook
- Learning Neo4j ebook
- Apresentação: De Relacional a (Grande) Grafo
- Webinar: do RDBMS para os Grafos
- Webinar: Relacional ao Grafo: Modelagem de Dados
- Webinar: Relacional ao Grafo: Importando Dados para o Neo4j
- Treinamento on-line: Começando com Neo4j
- Treinamentos em sala de aula

Neo4j, Inc. existe para ajudar o mundo a ter conhecimento dos dados. É por isso que a Neo4j, Inc. criou o Neo4j, o principal banco de dados orientado à grafos do mundo. Organizações em todo o mundo usam o Neo4j para extrair informações em tempo real, não só de seus dados, mas também de relacionamentos de dados. A Neo4j trouxe o poder dos bancos de dados orientados à grafos para as organizações grandes e pequenas - de empresas como Walmart, eBay, UBS, Cisco, HP, Telenor e Lufthansa. Esses negócios usaram o Neo4j para criar soluções tão variadas como motores de recomendação personalizados, modelos de gerenciamento de identidade e acesso, redes sociais, ferramentas de monitoramento de rede e aplicativos de gerenciamento de dados mestre.

Para maiores informações sobre Neo4j, contate-nos via e-mail ou telefone:

1-855-636-4532
info@neo4j.com