

**Neo4j Graph Data Modeling Chapter#1**

TST



체크

**□General, □Restricted, □Top**

**Security**

TABLE OF CONTENTS

Graphs Are Everywhere 3

1. Graphs in mathematics 3

2. The property graph model 4

3. Storage – native graph storage versus non-native graph storage 6

4. Reasons to use graph databases 7

5. What to use a graph database for 8

6. Choosing Neo4j for exploring graph databases 9

7. The structure of the book 10

# Graphs Are Everywhere

그래프는 우리 주변 어디에나 있다.

인터넷에 액세스할 때마다 데이터 패킷은 라우터, 스위치, 케이블 네트워크를 통해 이동하고 우리가 요청한 것을 전달한다. 문제의 주요 개념/개체(objects)를 나타내고 관련된 개념/개체 간의 관계나 상호 작용을 정의하는 동안, 일반적으로 개체를 나타내는 버블이나 상자, 상호 작용 이나 관계를 나타내기 위해 객체 사이에 화살표를 그린다.

우리는 다른 사람에게 경로를 설명하기 위해 지도를 그리면서 비슷한 표기법을 사용하기도 한다.

버블 과 화살표와 같은 이러한 표기법의 강점은 표현력에 있는데, 이런 표현력은 일반적으로 모델을 레코드와 테이블로 변경 할 때 손실되는 내용들이다.

그래프는 정보를 발견하고, 모델링의 어려움을 덜어주어 결론적으로 우리의 작업을 더 매끄럽게 만든다.

그래프를 더 잘 사용하려면 그래프 데이터베이스와 관련된 몇 가지 기본 개념을 이해해야 한다.

이 장에서 다음 사항에 대하여 살펴 볼 것이다 :

• Graphs in mathematics (수학에서 그래프)

• Tye property graph model (속성 그래프 모델)

• Reasons for using a graph database (그래프 데이터베이스를 사용하는 이유)

• Usage of graphs (그래프 사용 - 일부는 분명하고 일부는 명확하지 않은 그래프 문제)

• Advantages of using Neo4j (Neo4j 사용의 장점)

이 책에서 그래프 데이터 모델링을 설명하기 위해 Neo4j를 선택했으나, 여기에서 설명하는 모델링 개념은 모든 그래프 데이터베이스에 적용된다.

몇몇 독자는 Neo4j 사용자 경험이 있을 수 있어 이 장을 건너뛰고 싶을 것이다.

그러나 Neo4j를 처음 사용하거나 간단한 복습을 원하신다면 계속 진행이 필요하다.

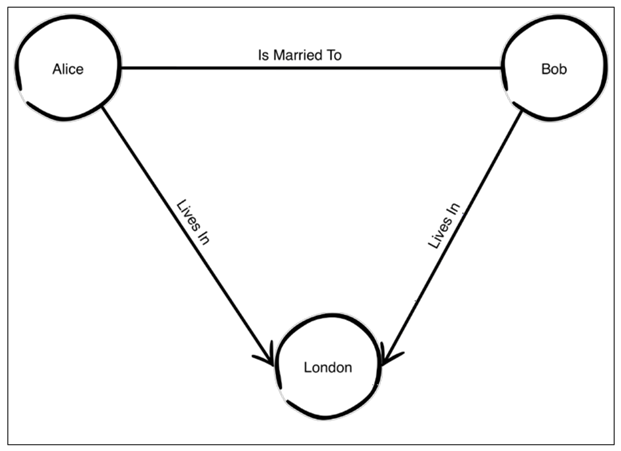
# Graphs in mathematics

그래프는 개체(Objects) 쌍이 링크(link)로 연결된 개체(Objects)들의 수학적 구조이다.

개체(Objects) 는 Node(Vertex)라고 하는 추상화로 표시되고, 링크(link)는 Relationship(edge)로 표시된다.

Relationship 는 하나의 특정 방향으로 의미론적 의미를 부여하며 향할 수 있다.

의미 체계가 양방향으로 작동하는 경우 링크를 표시하기 위해 무방향 관계를 사용할 수 있다.



<Figure 1.1 : Edges, vertices, directionality>

그림 1.1에는 노드로 표시되는 Alice, Bob 및 London의 세 가지 행위자(actor 또는 entity라 표현)가 있고, 그들 사이의 링크는 relationship 으로 표시된다.

"Alice는 Bob과 결혼했고 Bob은 Alice와 결혼했다" 이 경우 양 방향 사실이므로 "Is Married To" 를 방향이 없는 관계로 나타낸다.

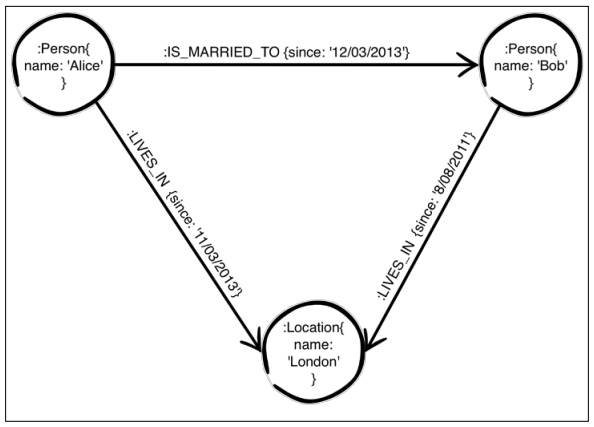
반면, Alice가 London에 산다는 것 맞으나, London이 Alice 에 산다는 문장은 적합하지 않으므로 방향성이 생기고 Alice에서 London으로의 직접적인 관계인 "Lives In"으로 표현된다.

# The property graph model

Neo4j에서는 속성 그래프 모델을 사용한다.

Property Graph Model은 수학의 그래프를 확장한 것이다.

다음 그림은 그림 1.1의 데이터가 Neo4j에서 표현될 수 있는 방법의 예를 보여준다 :



<Figure 1.2 : Nodes, relationships and properties>

앞의 그림 속성 그래프를 모델링하는 데 사용한 개념은 아래와 같다.

• Node: 엔티티(Entity)는 노드(Node)로 모델링된다. 그림 1.2에서 London, Bob, Alice는

모두 엔터티(Entity)다.

        • Label: 해당 도메인에서 Node의 역할을 나타내는 데 사용된다.

         Node는 동시에 여러 Label을 가질 수 있다.

         Label은 Node에 많은 의미를 부여 할수 있다.

         특정 Label에 한정된 제약 조건과 인덱스를 추가하는 데 사용된다.

         앞의 그림에서 :Person 및 :Location은 우리가 사용한 두 개의 Label 이다.

         이러한 각 Label에 대한 name 에 인덱스, 제약 조건을 추가할 수 있으며,

         결과적으로 두 개의 별도 인덱스가 생성된다.

         하나는 :Location용이고 다른 하나는 :Person용 이다.

        • Relationship: 두 노드 사이의 방향이 표시되고, 의미적으로 관련된 연결을 나타낸다.

         Neo4j의 Relationship에는 항상 Start Node, End Node, Relationship type 이 있다.

         방향으로 관계를 생성해야 하지만, 순회(traversing) 하는 동안 방향은 무시할 수

있다.

         그림 1.2의 :LIVES\_IN과 :IS\_MARRIED\_TO는 Relationship type 이다.

        • Property: node, relationship 에 대한 정보를 포함하는 키-값 쌍이다.

        이전 그림에서 name 및 since는 node, relationship에 대한 추가 정보를 제공하는

속성(property) 이다.

        Neo4j는 날짜, 문자열, 이중구조, 배열등 어떤 JVM(Java Virtual Machine)

type 도 속성(property)으로 사용 할 수 있다.

위 그래프 모델은 더 단순하고 표현력이 뛰어나며, 또한 명시적으로 관계를 호출한다.

외래 키(FK Key)를 사용하여 관계를 암시적으로 표시하는 RDBMS 와 달리, 관계를 명시적으로 정의하면 필요한 정보를 찾기 위해 관계를 순회(traversing)하여 데이터를 검색할 수 있다.

즉, 이것은 관련 데이터를 찾기 위해 인덱스 조회, 조인에 의존하기보다는 데이터의 연결성을 사용하는 실제적인 알고리즘 접근 방식이며, 또한 명시적 relationship(관계)는 property graph model 이 상호 연결되어 있기 때문에 대부분의 문제 영역에 자연스럽게 적합하도록 만든다.

# Storage – native graph storage versus non-native graph storage

모든 데이터베이스 관리 시스템과 마찬가지로 그래프 데이터베이스도 연결된 데이터에 대한 지속성(persistence)과 쿼리를 처리하는 스토리지, 쿼리 엔진의 개념이 있다.

쿼리 엔진은 쿼리 실행 과 데이터 검색/수정을 담당하며, Create, Read, Update, Delete 작업(CRUD)을 통해 그래프 데이터 모델을 보여준다.

스토리지는 데이터가 물리적으로 저장되는 방식과 검색 시 데이터가 논리적으로 표시되는 방식을 다룬다.

Storage 에 대한 지식은 그래프 데이터베이스를 선택 할 때 도움이 될 수 있다.

관계(Relationships)는 모든 도메인 모델의 중요한 부분이며 자주 탐색(to be traversed)되어야 한다.

그래프 데이터베이스에서 관계(relationships)는 추론(inferred)되기보다는 명시적(explicit)이다.

non-native 그래프 저장소(예: RDBMS, NoSQL- key/value DB, Document DB)에서는 작동하는 쿼리 엔진을 통해 관계(relationship)를 명시적(explicit)으로 만들고, native 그래프 저장소는 자체적으로 명시적으로 사용한다.

즉 non-native 그래프 저장소에 의존하는 그래프 데이터베이스에서 관계(relationship) 는 런타임(runtime) 시점에 추론 되어야 한다.

예를 들어, RDBMS에서 그래프를 모델링하려면 처리 엔진이 외래 키를 사용하여 관계(relationship)를 추론(infer)하고 런타임(runtime) 시점에 관계(relationship)를 구체화해야 한다.

이 문제는 계산 비용이 많이 들고 관련된 재귀 조인 때문에 여러 관계를 순회(traversing)할 경우 실행 불가능 할 수 있다.

HDFS와 같은 NoSQL 저장소, Cassandra와 같은 열 저장소 또는 Document 저장소를 사용하여 데이터를 저장하고 그래프 API를 노출하는 다른 그래프 데이터베이스가 있다.

NoSQL 저장소를 사용하는 그래프 데이터베이스에는 조인이 없지만 데이터베이스는 여전히 인덱스 조회를 사용해야 한다.

결론적으로 non-native 스토리지가 사용되는 경우 쿼리 엔진은 더 많은 계산 작업을 수행해야 한다.

반면 Neo4j는 native graph 스토리지를 사용한다.

각 node는 연결되는 관계(outgoing relationships)에 대한 핸들을 가지고 있고, 각 관계(relationship)는 차례로 다음 node(terminal node)를 알고 있다.

Neo4j는 런타임시 이웃 node를 찾기 위해 인덱스 조회를 수행할 필요가 없다.

대신 현재 노드(node)의 관계(relationship)를 보고(looking) 인접 노드를 식별할 수 있다.

이 기능을 index-free adjacency(인덱스 없는 인접성) 이라고 하며, Index-free adjacency(인덱스 없는 인접성) 은 그래프를 탐색하는 동안 Neo4j 쿼리 엔진이 의미있는 성능 향상을 가질 수 있도록 한다.

# Reasons to use graph databases

매일 아침 우리가 Facebook feed를 확인할 때, 우리는 친구들과 뉴스의 연이은 업데이트 소식을 알 수 있다. 데이터가 어떻게 연결되어 있는지에 대한 정보를 사용하고 그것을 우리의 개별 선호도와 일치시킴으로써, Facebook은 우리의 네트워크로부터 우리에게 중요하고 흥미로운 일련의 활동들을 구축한다.

Linked in은 우리 네트워크 내에서 일을 제안하면서 비슷한 일을 한다.

Google 지도나 TomTom이나 Sygic 지도와 같은 일부 애플리케이션을 가동하여 목적지로 이동하기 시작하면 도시 내 다양한 교차로의 연결을 나타내는 데이터를 사용하여 최적의 통과 방법을 알아낸다.

온라인 쇼핑을 하는 동안 우리가 이미 구매한 제품이나 다른 사람들이 구매한 유사한 제품과 얼마나 밀접하게 연결되어 있는지를 기반으로 제품을 추천한다.

이렇듯, 우리는 매일 점점 더 많은 연결된 데이터를 자각하지 못한 채 활용하고 있다.

연결된 데이터를 처리할 때 그래프 데이터베이스는 다음과 같은 이점을 제공한다.

        • 그래프 데이터베이스의 쿼리 성능은 RDBMS나 다른 NoSQL 대안보다 몇 배 더 우수하다.

        데이터 세트가 증가함에 따라 RDBMS 조인 성능은 지속적으로 증가하는 조인 테이블 크기로 인해

저하된다.

        반면에 그래프 순회(traversal) 는 그래프의 일부에 국한(localized) 된다.

        따라서 쿼리 실행 시간은 저장된 전체 데이터 양에 비례하지 않고 방문한 노드 수에 비례하며,

        데이터가 기하급수적으로 증가하더라도 쿼리 성능은 시간이 지남에 따라 상당히 일정하게

유지된다.

        • 유연성(flexibility)과 민첩성(agility)은 비즈니스 요구가 끊임없이 변화하는 오늘날의

환경에서 주요 고려 사항이다.

         개발자는 코딩을 시작하기 전에 데이터 모델을 잠그지 않고(locking down) 모델을 점진적으로

생각할 수 있는 도구가 필요하다.

그래프 데이터베이스를 사용하면 기존 쿼리를 변경하지 않고도 관계, 노드 유형 및 속성을

추가할 수 있다.

        모델을 점진적으로 연결하여 보다 정교한 질의를 할 수 있다.

        이러한 유연성 덕분에 마이그레이션도 줄어든다.

        데이터 모델 변경 시에도 비교적 부담이 없고 오랜 시간 데이터베이스를 오프라인으로 전환하지

않고도 마이그레이션이 가능해 개발팀(teams)이 인프라와 통신을 관리하는 대신 도메인에

집중하면서 소프트웨어를 더 빠르게 제공할 수 있다.

        • 모호성(ambiguity)이 낮으면 더 나은 모델이 생성 된다.

      그래프 데이터베이스는 스키마가 없기 때문에 스키마는 응용 프로그램에 의해 결정되므로 더

잘 검증된다.

        테이블에 저장되는 방식에 비해 도메인 모델의 모호성이 없기 때문에 개발자가 더 나은 디자인

사고를 할 수 있다.

        • 디자인에서 납기까지의 시간이 단축된다.

         개발자의 관점에서 그래프 데이터베이스의 가장 큰 특징 중 하나는 화이트보드 친화적이라는

것이다.

        화이트보드에 데이터 모델을 만들 수 있으며, 그 데이터 모델을 일련의 테이블로 변환 해야

한다고 걱정할 필요가 없다.

        이를 통해 개발자들은 모델 변환(translation) 보다는 개발에 집중할 수 있어 시간을 절약할 수

있다.

지금까지 언급된 모든 내용이 기술적 내용처럼 보일 수 있으나, 사실 경제 문제(economics)로 귀결된다.

그래프 데이터베이스는 데이터가 고도로 연결되어 있을 때 사용하는 것이 보다 경제적(economic sense) 이다.

# What to use a graph database for

먼저 그래프 데이터베이스에 더 적합한 몇 가지 문제 설명을 인용하겠다.

라우팅은 그래프 문제이며 이러한 측면에서 많은 연구가 수행되었다.

세계 최고의 배달 서비스 중 한곳은 Neo4j 기반 솔루션을 사용하여 전 세계에서 수집되는 정보를 기반으로 실시간으로 패키지를 라우팅한다.

소셜 네트워크는 사용자의 연결을 활용하여 데이터를 가져오고 액세스 가능한 항목과 그렇지 않은 항목을 결정하기 때문에 그래프에 적합한 문제이다.

특히 Facebook은 그래프 검색을 사용하여 사용자에게 더 나은 검색을 할 수 있도록 노출했다.

Facebook은 feed를 선별하기 위해 사람과 친구의 그래프에 크게 의존한다.

추천은 다시 그래프 데이터베이스를 사용하여 해결할 수 있는 그래프 문제이다.

eBay는 과거 MySQL를 사용했지만 현재 Neo4j를 사용하고 있다.

라우팅, 소셜 네트워크, 추천 모두 명백한 그래프 문제이지만, 기업들은 최근 데이터를 그래프에 끼워 맞춤으로써 수많은 문제를 해결했다.

예를 들어, 검색은 그래프 문제로 나타나지 않으며 매우 직관적인 문제도 아니다.

그러나 Google은 지식 그래프를 사용하여 검색 중인 용어와 컨텐츠가 얼마나 잘 연결되어 있는지에 따라 검색 결과를 제공한다.

더 최근에는 Facebook이 소셜 그래프를 활용하여 검색이 더 잘 되도록 돕고 있다.

의학 연구는 그래프가 사용되는 또 다른 영역이다.

의료 데이터는 고도로 상호 연결되어 있으므로 그래프 데이터베이스를 사용하면 큰 이점을 얻을 수 있다.

기업들은 이제 약물 발견 및 의료 정보 저장을 위해 그래프 데이터베이스를 사용하고 있다.

온톨로지의 스토리지는 머신 러닝 및 분석에서 애플리케이션을 빠르게 찾는 그래프 데이터베이스를 사용하여 점점 더 많이 해결되고 있다.

기업들은 또한 에너지 공급 및 운송과 같은 영역에서 그래프 데이터베이스를 사용하고 있다.

# Choosing Neo4j for exploring graph databases

Neo4j는 ACID(Atomicity, Consistency, Isolation, Durability) 특성을 충족하는 빠른 native 그래프 데이터베이스이다.

트랜잭션사용을 통해 개발자는 트랜잭션 실패가 데이터베이스의 상태를 변경하지 않은 상태로 유지하여 원자성(Atomicity) 을 보장할 수 있다.

데이터베이스를 변경해도 데이터가 손상되지 않으므로 일관성(Consistency) 을 보장한다.

트랜잭션에 의해 수정된 데이터는 커밋될 때까지 다른 트랜잭션과 격리(Isolation)된다.

Neo4j는 영구 그래프 데이터베이스이므로 커밋된 트랜잭션의 결과를 항상 검색할 수 있는 내구성(Durable)을 가진다.

TinkerPop 스택을 지원하기 시작했다.

TinkerPop 스택에 대한 자세한 내용은 [http://www.tinkerpop.com](http://www.tinkerpop.com/)에서 확인할 수 있다.

Neo4j는 다양한 모델링 및 기술 지원을 제공하며,

다음과 같은 실제 시스템을 구축할 때 이는 매우 중요하다:

        • Neo4j는 가장 성숙한 그래프 데이터베이스로 2003년부터 24시간 운영되어 왔다.

         Neo4j는 거대한 커뮤니티를 가진 오픈 소스이다.

         Neo4j 개발팀은 해당 커뮤니티와 연계되어 있어 기능과 버그가 빠르게 해결한다.

      Neo4j는 엔진이 native graph 처리를 수행할 수 있는 native graph 저장소를 제공한다.

        쿼리 언어에서 디스크에 이르기까지 모든 것이 트랜잭션 저장 및 그래프 데이터의 신속한 검색

에 기계적으로 적합하다.

        • Cypher는 Neo4j에서 데이터를 검색하는 데 사용되는 매우 표현력이 뛰어난 쿼리 언어이다.

         일부 측면에서 SQL과 표면적으로 유사하지만, Cypher는 성능이 뛰어난 그래프 쿼리와 쓰기를

위해 구축된 유일한 선언적 쿼리 언어이다.

         Neo4j Java API는 JVM 기반 언어에서 보다 명령적이고 성능이 뛰어난 쿼리 방법으로 사용할 수

있다. 이는 명령형 및 선언적 쿼리를 지원함으로써 두 가지 모두의 장점을 제공한다.

(Neo4j는 장기적으로 Gremlin 지원에서 벗어날 계획이며 현재 Gremlin은 플러그인을 통해 지원

한다.)

Neo4j는 오픈 소스로 플러그인이 기능을 강화하거나 추가할 수 있으며 핵심 데이터베이스를

중심으로 도구 생태계가 활발하다

        • 그래프를 업데이트하는 모든 Cypher 문은 트랜잭션 내에서 실행된다.

     트랜잭션이 존재하면 새로 실행된 Cypher 쿼리가 트랜잭션에서 실행된다.

        트랜잭션이 없으면 명령문 자체가 트랜잭션이 된다.

        • 커뮤니티의 활성화는 놀랍다.

      이것은 또한 프로젝트가 오픈 소스이기 때문에 부분적으로 가능한 일이다.

         Neo4j는 현재 UBS, Cisco, Walmart, eBay, Telenor, HP, Pitney Bowes, Accenture, Lockheed

Martin, Glassdoor 등 여러 회사에서 프로덕션에 사용하고 있다.

# The structure of the book

이 책은 두 부분으로 나누어져 있다.

• 섹션 1 (2장 Flights 와 Cities 모델링 ~ 5장 데이터 모델 리팩토링)

* 일상에서 사용할 그래프 모델링 개념을 이해하는 데 필수적 모델
* 그래프를 모델링하는 방법, 쿼리하는 방법, 도메인의 변경 사항을 수용하기 위해 그래프 데이터베이스를 발전시키는 방법, RDBMS 데이터 모델을 그래프 디자인으로 변환하는 방법을 다룸

• 섹션 2 (6장 Communication 체인 모델링 ~ 8장 추천 및 과거 데이터 분석)

* 최적화 또는 특수한 경우에 필요할 수 있는 참조 모델.
* 모델링 체인 및 모델링 체인의 장점, 액세스 제어 모델링, 존재하는 데이터를 기반으로 한 추천 시스템 설계.

|  |  |  |
| --- | --- | --- |
| 챕터 | 제목 | 챕터 내용 (Draft) |
| 1 | Graphs Are Everywhere | • 수학에서 그래프 란? • 그래프 모델 구성요소 • 그래프 데이터베이스를 사용하는 이유 와 용도 • 책 구성 설명 |
| 2 | Modeling Fights and Cities | • 여행 일정내 Flight & City 실체 모델링 시각화 • Cypher 를 이용한 식별된 실체 node, label, property, relationship, unique constraint, index 추가 |
| 3 | Formulation and Itinerary | • 가변 relationship 쿼리 (여러 홉 경로 쿼리) • Node 집합 데이터 쿼리 |
| 4 | Modeling Bookings and Users | • Booking Modeling ( RDBMS ) • Booking Modeling ( Graph) • Cypher 를 이용한 예약 및 여행 찾기 |
| 5 | Refactoring the Data Model | • 추가 비즈니스 요구 사항 반영 Graph Model 수정 |
| 6 | Modeling Communication Chains | • Review 모델링 • 시간 순 발생 데이터에 대한 모델링 |
| 7 | Modeling Access Control | • 그래프에서 접근 제어 구조 생성 • 접근 제어 구조를 이용한 데이터 쿼리 |
| 8 | Recommendations and Analysis of Historical Data | • 여행 도시 추천 • 과거 데이터를 사용한 패턴 발견 |
| 9 | Wrapping Up | 마무리 |