

Architecture of Enterprise Applications 16

Neo4J & Graph Computing

Haopeng Chen

REliable, INtelligent and Scalable Systems Group (REINS)

Shanghai Jiao Tong University

Shanghai, China

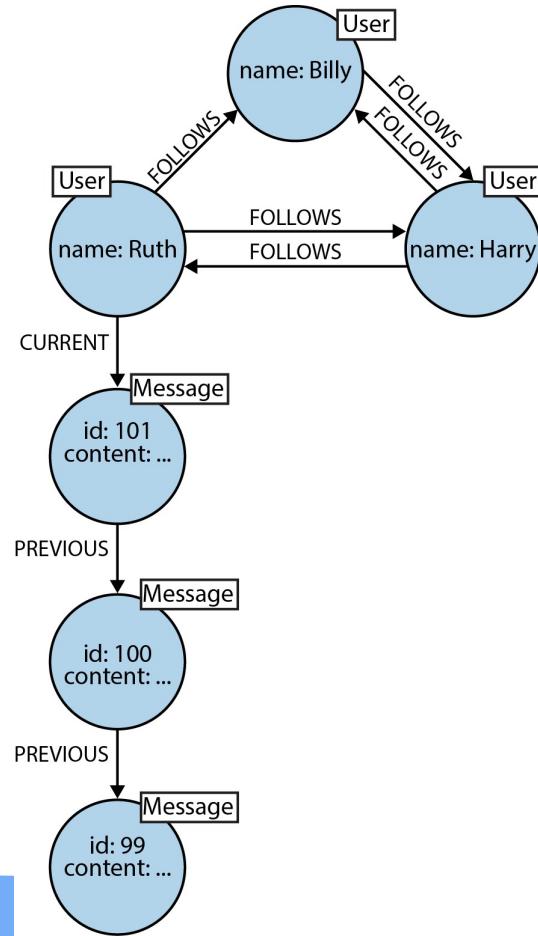
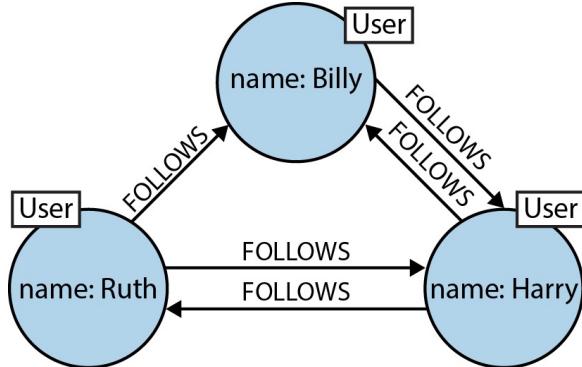
<http://reins.se.sjtu.edu.cn/~chenhp>

e-mail: chen-hp@sjtu.edu.cn

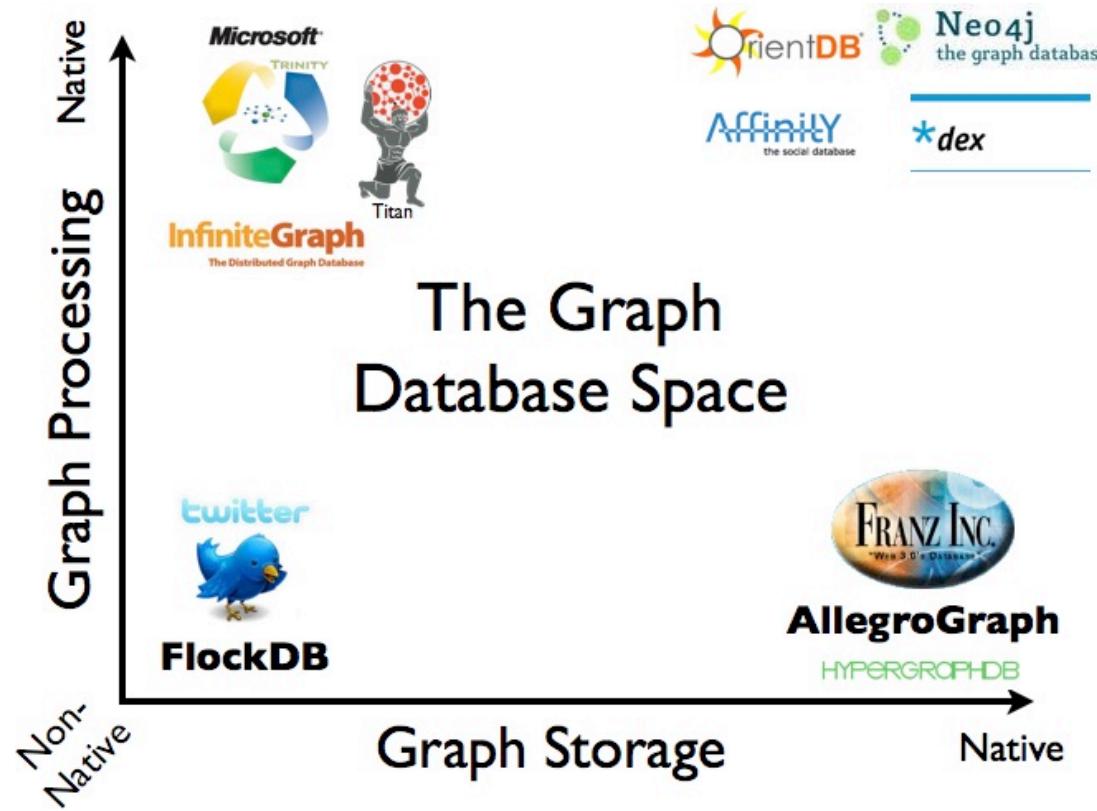
- **Contents**
 - Neo4J
 - Graph Database
 - Options for Storing Connected Data
 - Data Modeling with Graphs
 - Building a Graph Database Application
 - Graph Database Internals
- **Objectives**
 - 能够根据数据特性和数据访问模式，识别适合图数据库存储的数据，设计并实现其在图数据库中的存储和访问方案

What is a Graph?

- Formally, a graph is just a collection of *vertices* and *edges*
 - or, in less intimidating language, a set of *nodes* and the *relationships* that connect them.
 - Graphs represent entities as nodes and the ways in which those entities relate to the world as relationships.

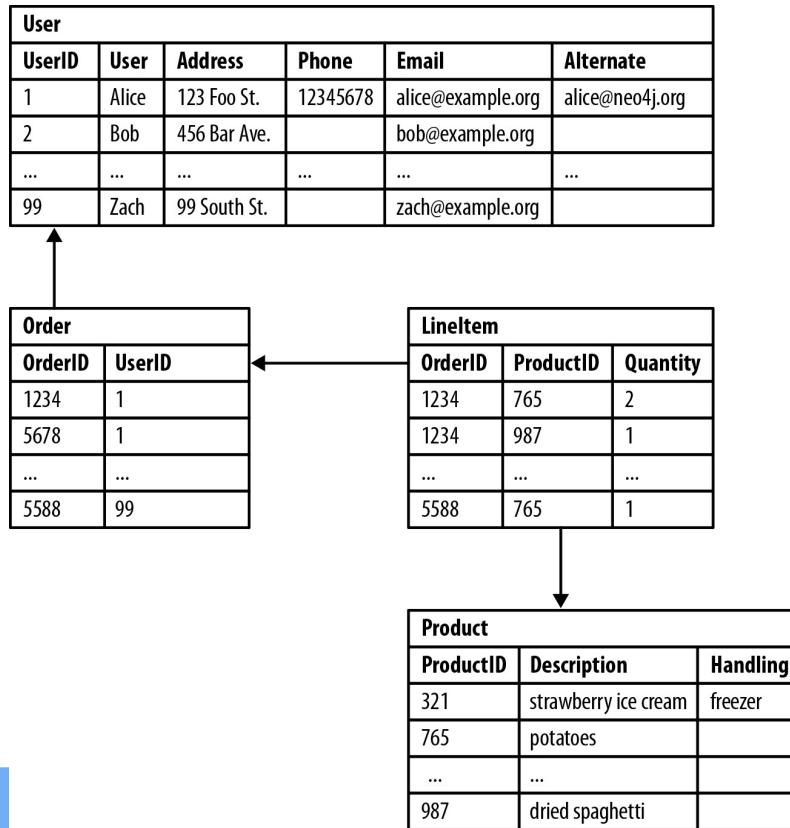


- A *graph database management system* (henceforth, a *graph database*) is
 - an online database management system with Create, Read, Update, and Delete (CRUD) methods that expose a graph data model.
 - Graph databases are generally built for use with transactional (OLTP) systems.
 - Accordingly, they are normally optimized for transactional performance, and engineered with transactional integrity and operational availability in mind.
- There are two properties of graph databases we should consider when investigating graph database technologies:
 - *The underlying storage*
 - *The processing engine*



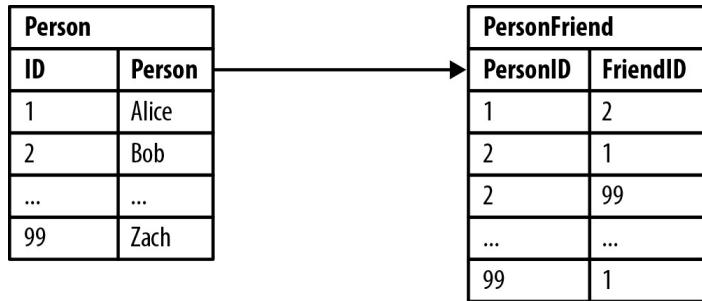
Options for Storing Connected Data

- Relational Databases Lack Relationships



Options for Storing Connected Data

- Relational Databases Lack Relationships



Example 3. Alice's friends-of-friends

```
SELECT p1.Person AS PERSON,  
       p2.Person AS FRIEND_OF_FRIEND  
  FROM PersonFriend pf1 JOIN Person p1  
    ON pf1.PersonID = p1.ID  
   JOIN PersonFriend pf2  
    ON pf2.PersonID = pf1.FriendID  
   JOIN Person p2  
    ON pf2.FriendID = p2.ID  
 WHERE p1.Person = 'Alice' AND pf2.FriendID <> p1.ID
```

Example 1. Bob's friends

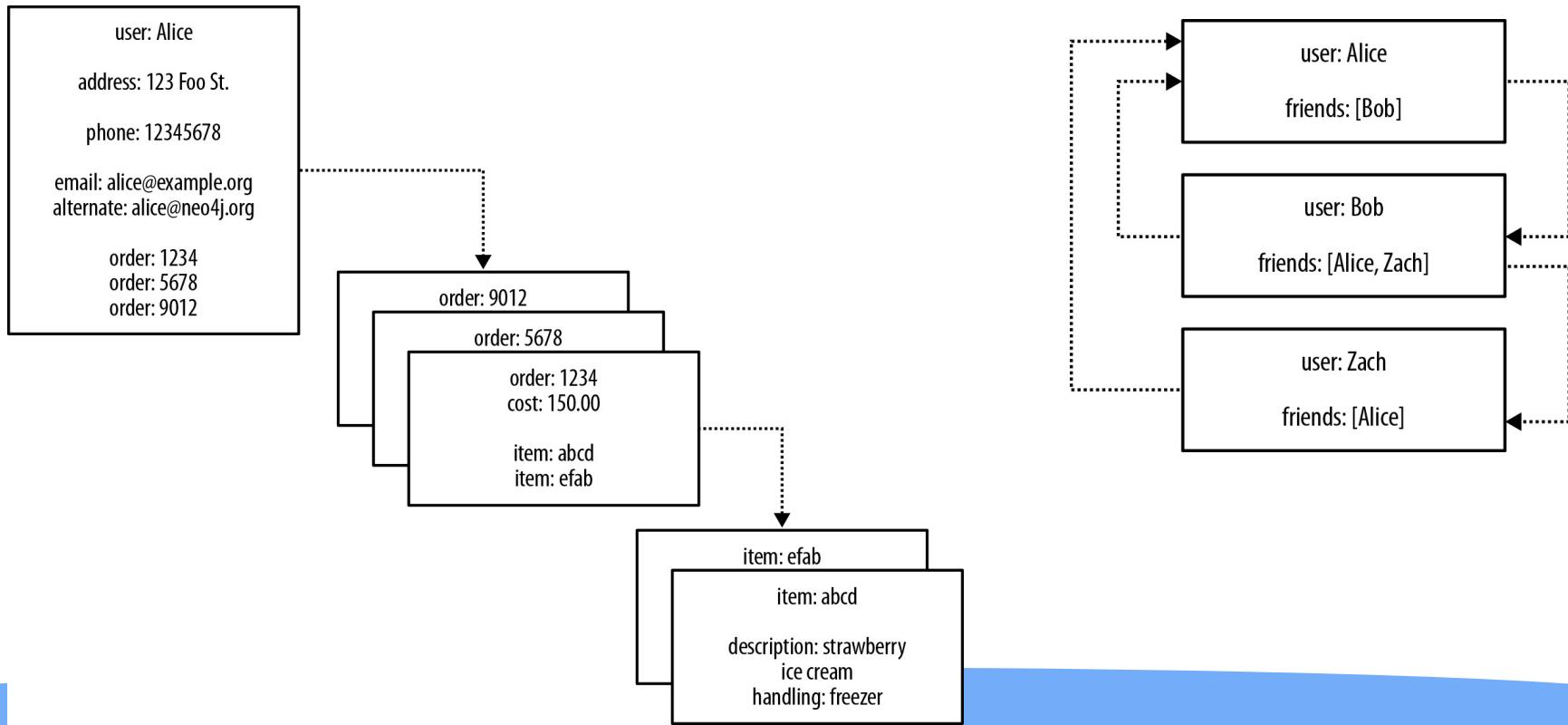
```
SELECT p1.Person  
  FROM Person p1 JOIN PersonFriend  
    ON PersonFriend.FriendID = p1.ID  
   JOIN Person p2  
    ON PersonFriend.PersonID = p2.ID  
 WHERE p2.Person = 'Bob'
```

Example 2-2. Who is friends with Bob?

```
SELECT p1.Person  
  FROM Person p1 JOIN PersonFriend  
    ON PersonFriend.PersonID = p1.ID  
   JOIN Person p2  
    ON PersonFriend.FriendID = p2.ID  
 WHERE p2.Person = 'Bob'
```

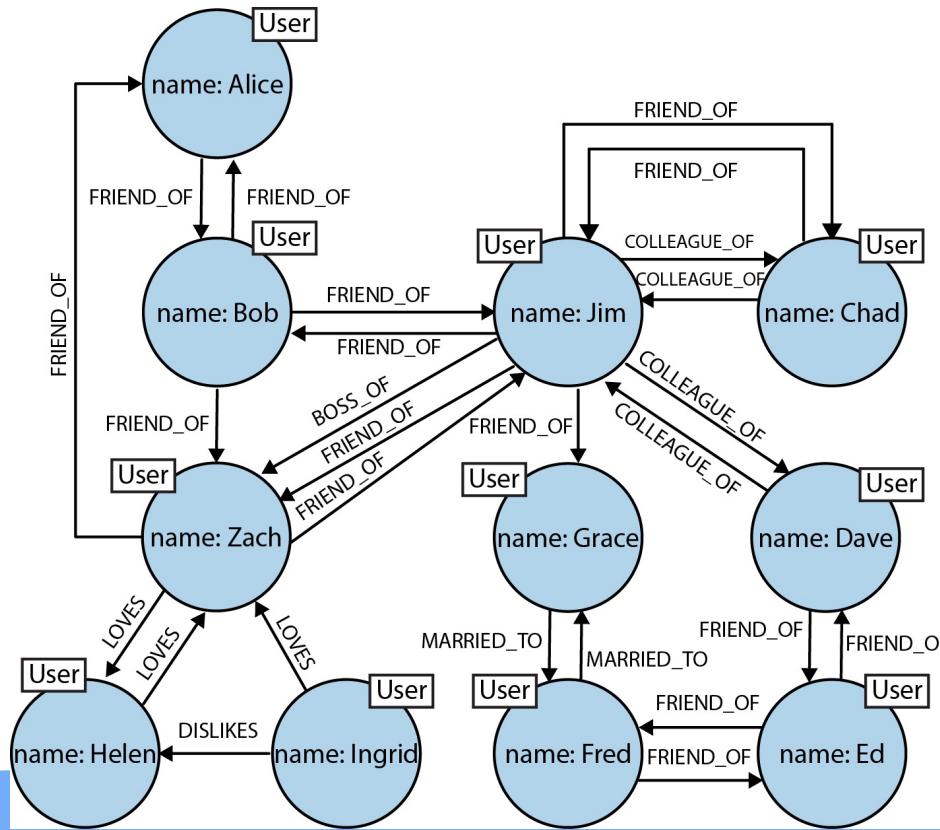
Options for Storing Connected Data

- **NOSQL Databases Also Lack Relationships**



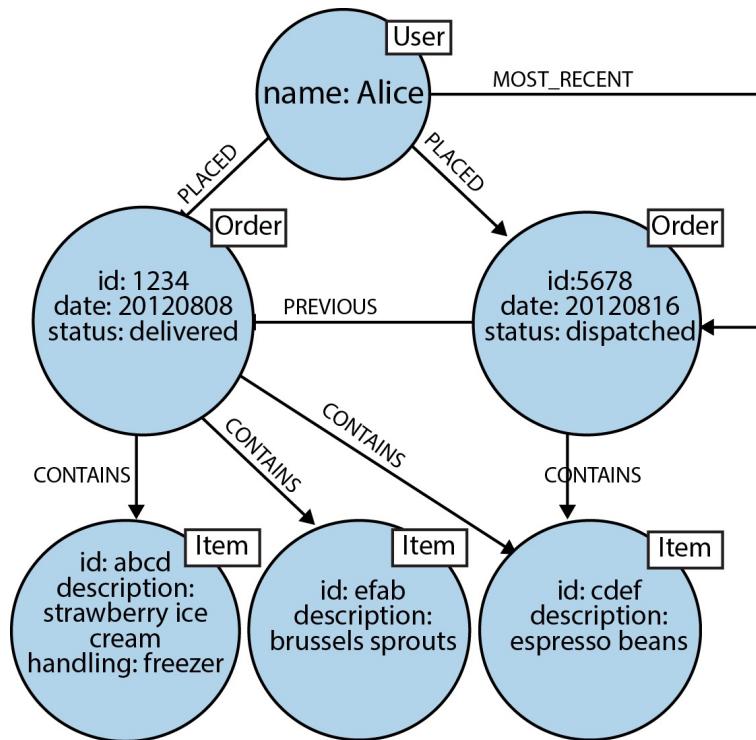
Options for Storing Connected Data

- **Graph Databases Embrace Relationships**



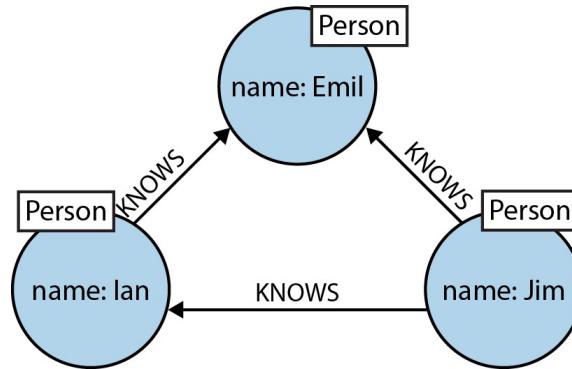
Options for Storing Connected Data

- Graph Databases Embrace Relationships



- **The Labeled Property Graph Model**
- A *labeled property graph* is made up of *nodes*, *relationships*, *properties*, and *labels*.
 - Nodes contain properties. Think of nodes as documents that store properties in the form of arbitrary key-value pairs. In Neo4j, the keys are strings and the values are the Java string and primitive data types, plus arrays of these types.
 - Nodes can be tagged with one or more labels. Labels group nodes together, and indicate the roles they play within the dataset.
 - Relationships connect nodes and structure the graph. A relationship always has a direction, a single name, and a *start node* and an *end node*—there are no dangling relationships. Together, a relationship's direction and name add semantic clarity to the structuring of nodes.

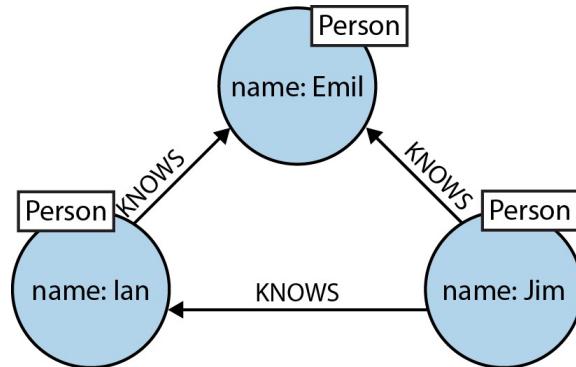
- **Querying Graphs: Cypher**
 - Cypher is an expressive (yet compact) graph database **query** language.



- Here's the equivalent ASCII art representation in Cypher:
`(emil:Person {name:'Emil'}) <-[:KNOWS]-(jim:Person {name:'Jim'}) -[:KNOWS]->(ian:Person {name:'Ian'}) -[:KNOWS]->(emil)`

Data Modeling with Graphs

- **Querying Graphs: Cypher**
 - Cypher is an expressive (yet compact) graph database **query** language.



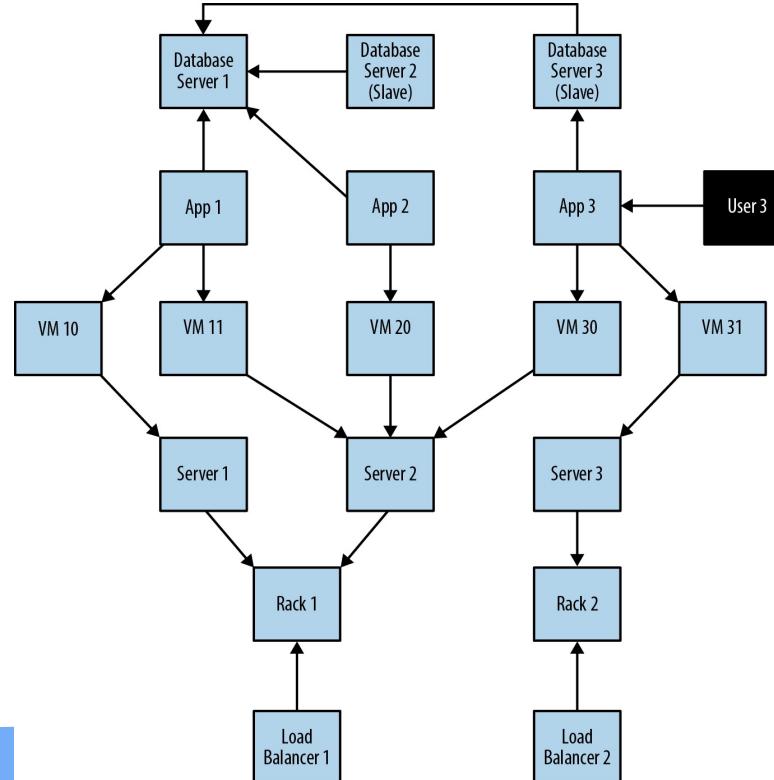
- Match:

```
MATCH (a:Person)-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c)  
WHERE a.name = 'Jim'  
RETURN b, c
```

- **Querying Graphs: Cypher**
- The other clauses we can use in a Cypher query include:
 - WHERE
 - Provides criteria for filtering pattern matching results.
 - CREATE and CREATE UNIQUE
 - Create nodes and relationships.
 - MERGE
 - Ensures that the supplied pattern exists in the graph, either by reusing existing nodes and relationships that match the supplied predicates, or by creating new nodes and relationships.
 - DELETE
 - Removes nodes, relationships, and properties.
 - SET
 - Sets property values
 - FOREACH
 - Performs an updating action for each element in a list.
 - UNION
 - Merges results from two or more queries.
 - WITH
 - Chains subsequent query parts and forwards results from one to the next. Similar to piping commands in Unix.
 - START
 - Specifies one or more explicit starting points—nodes or relationships—in the graph.

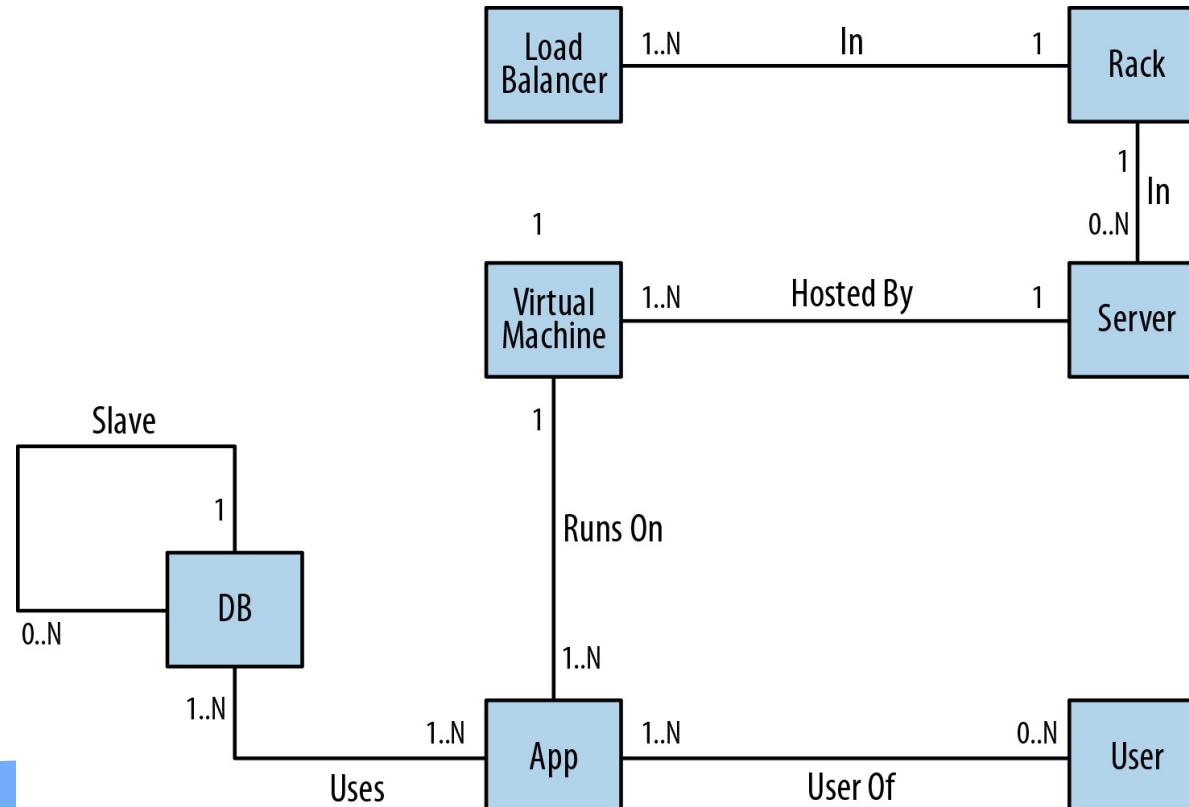
Data Modeling with Graphs

- **A Comparison of Relational and Graph Modeling**
 - To facilitate this comparison, we'll examine a simple data center management domain.



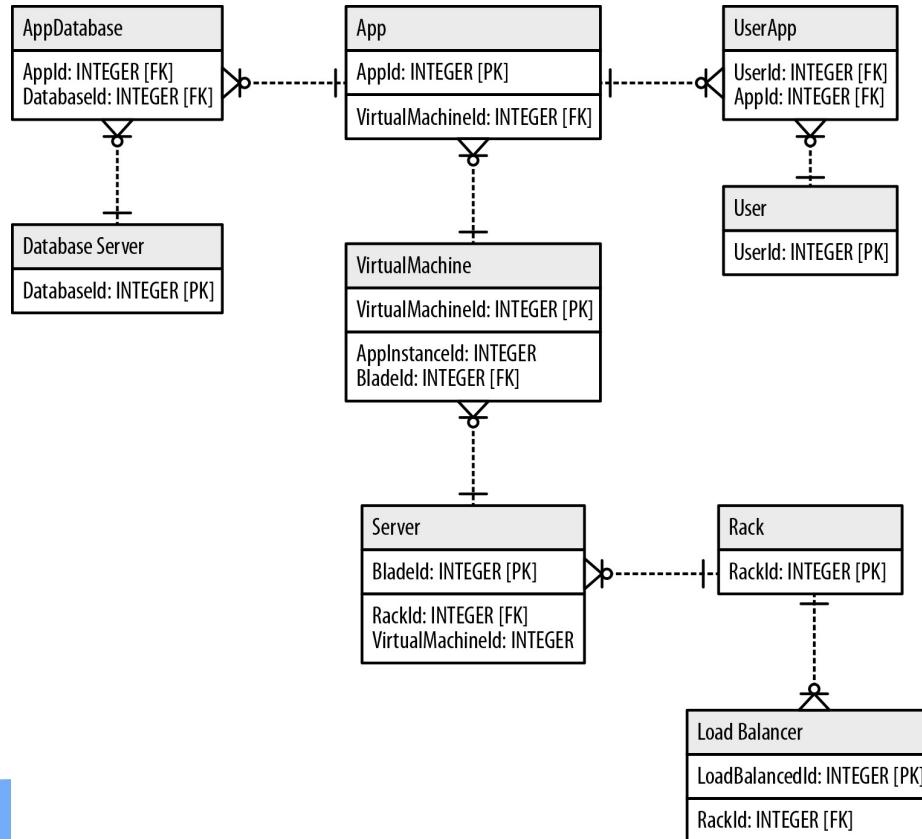
Data Modeling with Graphs

- **Relational Modeling in a Systems Management Domain**



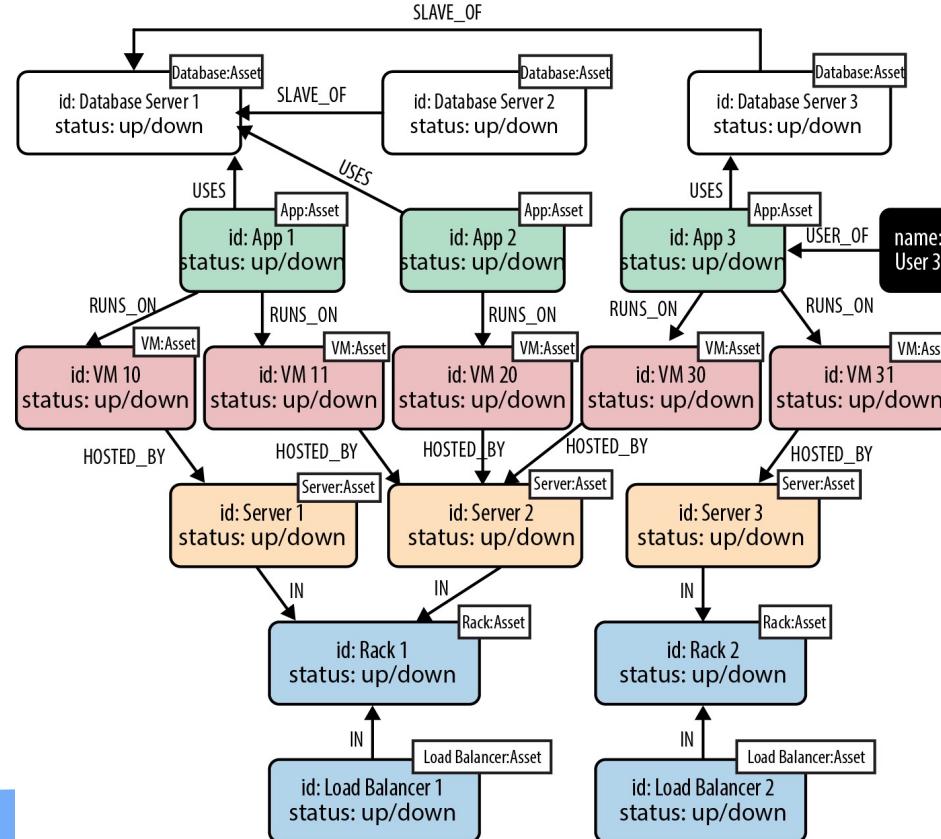
Data Modeling with Graphs

- **Relational Modeling in a Systems Management Domain**



Data Modeling with Graphs

- **Graph Modeling in a Systems Management Domain**



- **Test Graph Model**

- When a user reports a problem, we can limit the physical fault-finding to problematic network elements between the user and the application and the application and its dependencies.

- In our graph we can find the faulty equipment with the following query:

- **MATCH (user:User)-[*1..5]-(asset:Asset)
WHERE user.name = 'User 3' AND asset.status = 'down' RETURN DISTINCT asset**

- This allows us to match paths such as:

- (user)-[:USER_OF]->(app)

- (user)-[:USER_OF]->(app)-[:USES]->(database)

- (user)-[:USER_OF]->(app)-[:USES]->(database)-[:SLAVE_OF]->(another-database) (user)-[:USER_OF]->(app)-[:RUNS_ON]->(vm)

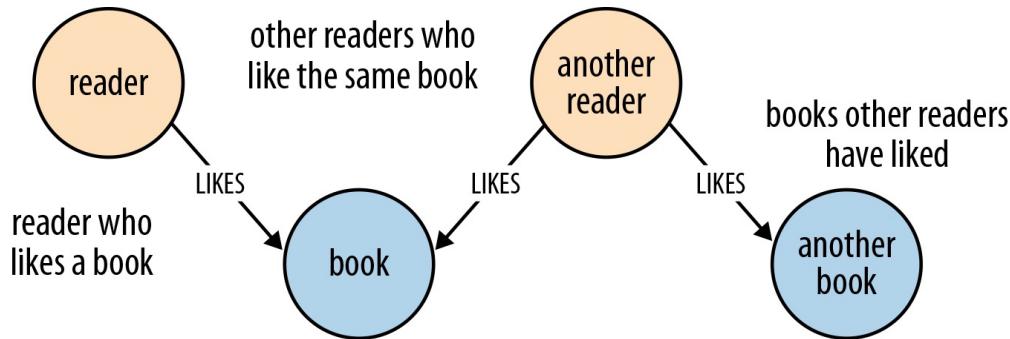
- (user)-[:USER_OF]->(app)-[:RUNS_ON]->(vm)-[:HOSTED_BY]->(server)

- (user)-[:USER_OF]->(app)-[:RUNS_ON]->(vm)-[:HOSTED_BY]->(server)-[:IN]->(rack)

- (user)-[:USER_OF]->(app)-[:RUNS_ON]->(vm)-[:HOSTED_BY]->(server)-[:IN]->(rack) <-[:IN]->(load-balancer)

Building a Graph Database Application

- **Describe the Model in Terms of the Application's Needs**
- Here's an example of a user story for a book review web application:
 - **AS A** reader who likes a book, **I WANT** to know which books other readers who like the same book have liked, **SO THAT** I can find other books to read.



- since *Alice likes Dune, find books that others who like Dune have enjoyed:*
`MATCH (:Reader {name:'Alice'})-[:LIKES]->(:Book {title:'Dune'})
<-[:LIKES]-(:Reader)-[:LIKES]->(books:Book)
RETURN books.title`

- **Nodes for Things, Relationships for Structure**

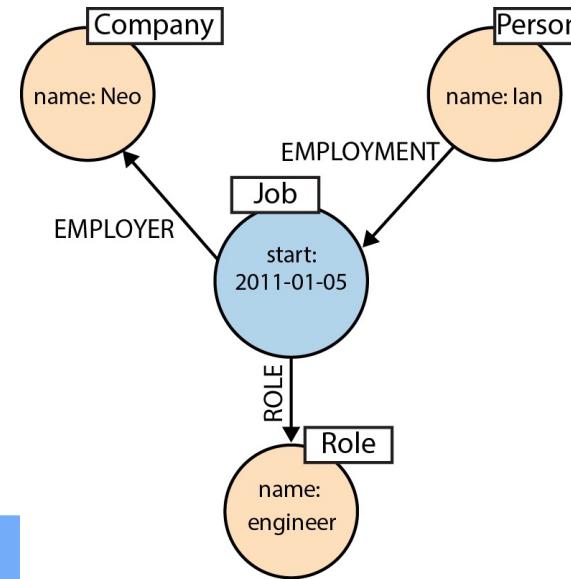
- Use nodes to represent entities—that is, the *things* in our domain that are of interest to us, and which can be labeled and grouped.
- Use relationships both to express the *connections* between entities and to establish semantic context for each entity, thereby structuring the domain.
- Use relationship direction to further clarify relationship semantics. Many relationships are asymmetrical, which is why relationships in a property graph are always directed. For bidirectional relationships, we should make our queries ignore direction, rather than using two relationships.
- Use node properties to represent entity *attributes*, plus any necessary entity metadata, such as timestamps, version numbers, etc.
- Use relationship properties to express the strength, weight, or quality of a relationship, plus any necessary relationship metadata, such as timestamps, version numbers, etc.

Building a Graph Database Application

- **Model Facts as Nodes**
- **Employment**

- The Figure shows how the fact of Ian being employed by Neo Technology in the role of engineer can be represented in the graph.

```
CREATE (:Person {name:'Ian'})-[:EMPLOYMENT]-> (employment:Job {start_date:'2011-01-05'}) -[:EMPLOYER]->(:Company {name:'Neo'}),  
(:employment)-[:ROLE]->(:Role {name:'engineer'})
```

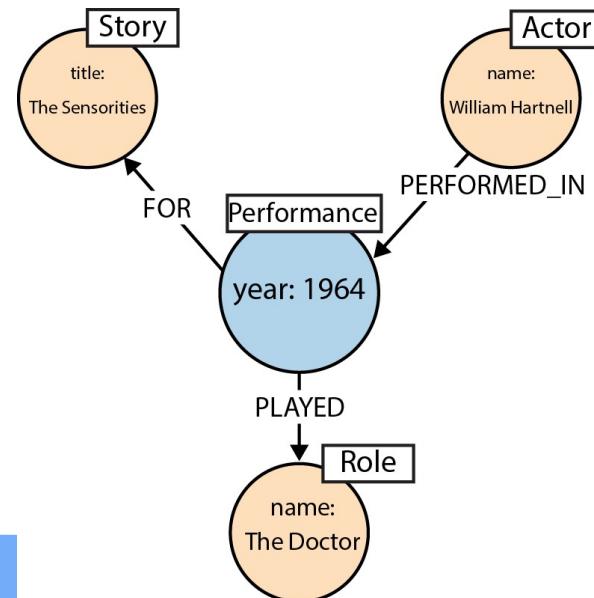


Building a Graph Database Application

- **Model Facts as Nodes**
- **Performance**

- The Figure shows how the fact that William Hartnell played The Doctor in the story *The Sensorites* can be represented in the graph.

```
CREATE (:Actor {name:'William Hartnell"})-[:PERFORMED_IN]-> (performance:Performance {year:1964})-[:PLAYED]-> (:Role {name:'The Doctor'}),  
(:performance)-[:FOR]->(:Story {title:'The Sensorites'})
```

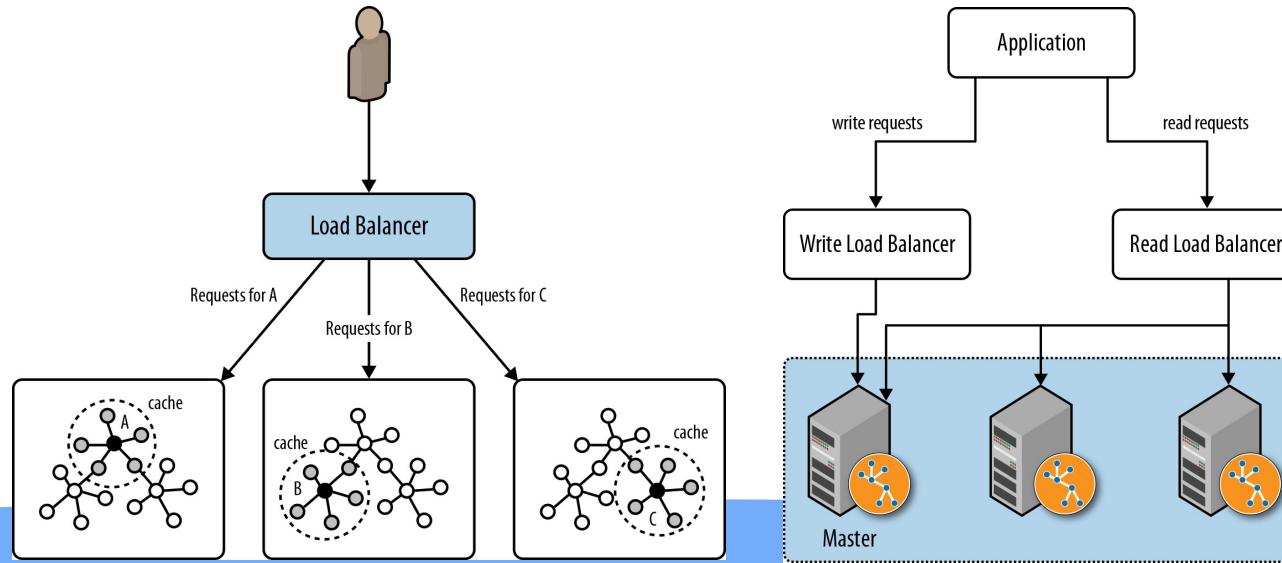


Building a Graph Database Application

- **Embedded versus Server**

- Most databases today run as a server that is accessed through a client library. Neo4j is somewhat unusual in that it can be run in embedded as well as server mode.
- In embedded mode, Neo4j runs in the same process as our application.
- Running Neo4j in server mode is the most common means of deploying the database today.

- **Cluster**

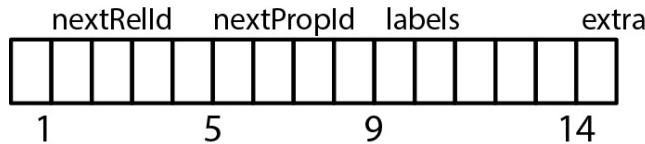


- Neo4j stores graph data in a number of different *store files*.
 - Each store file contains the data for a specific part of the graph (e.g., there are separate stores for nodes, relation- ships, labels, and properties).

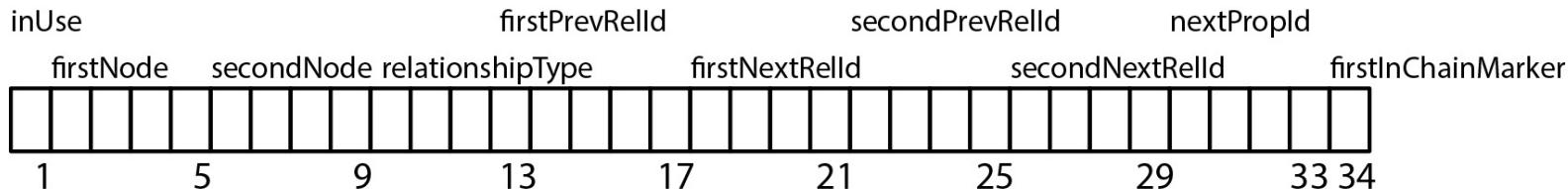
Node (15 bytes)



inUse

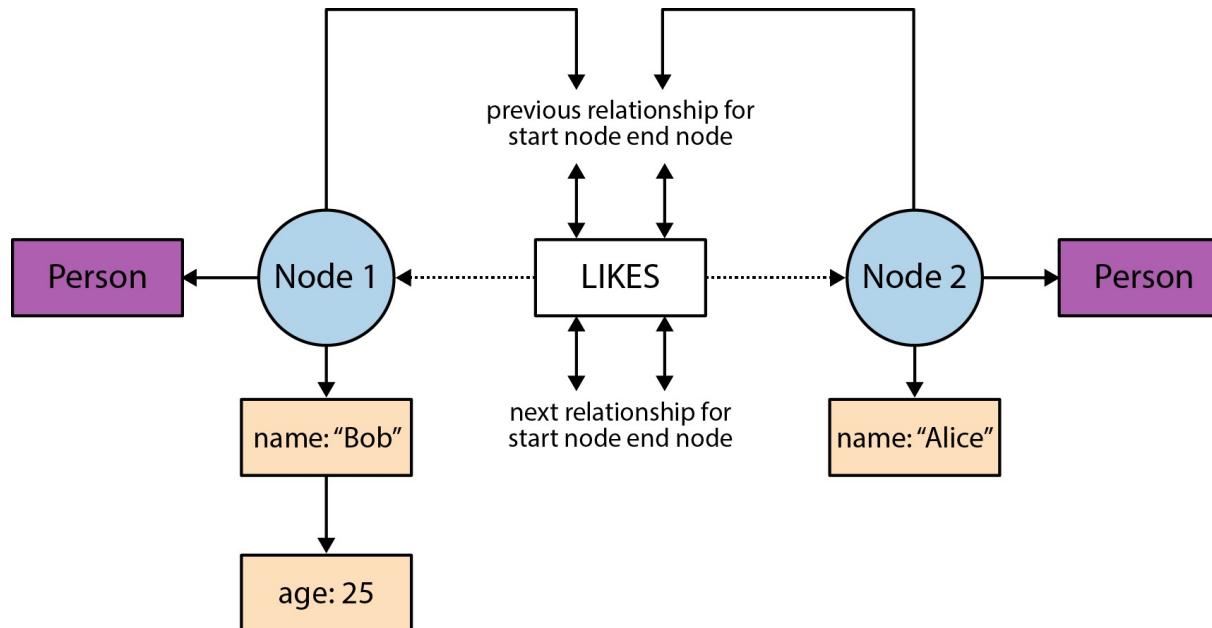


Relationship (34 bytes)



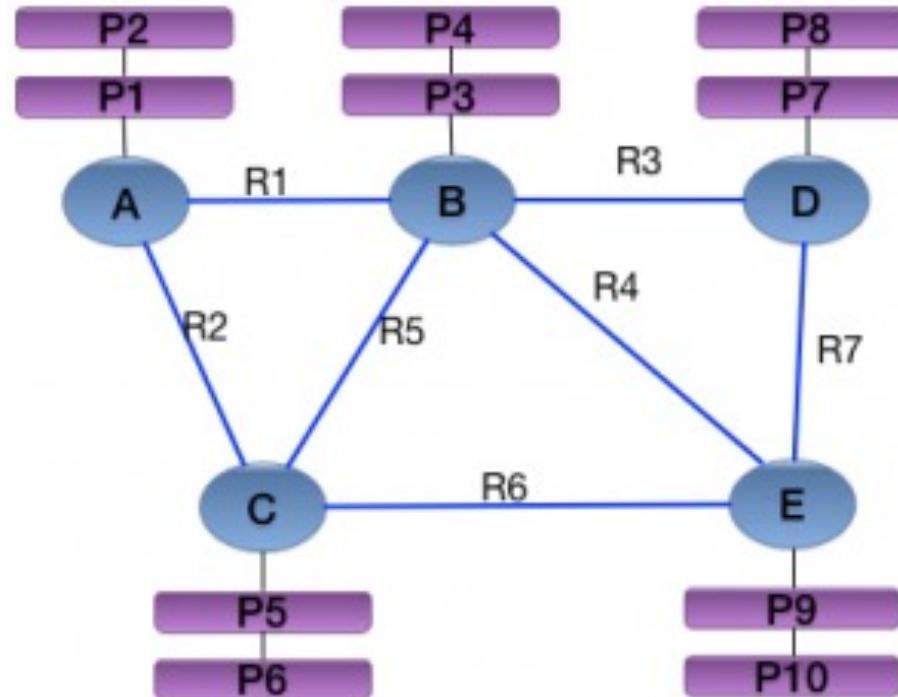
Graph Database Internals

- Neo4j stores graph data in a number of different *store files*.
 - Doubly Linked Lists in the Relationship Store



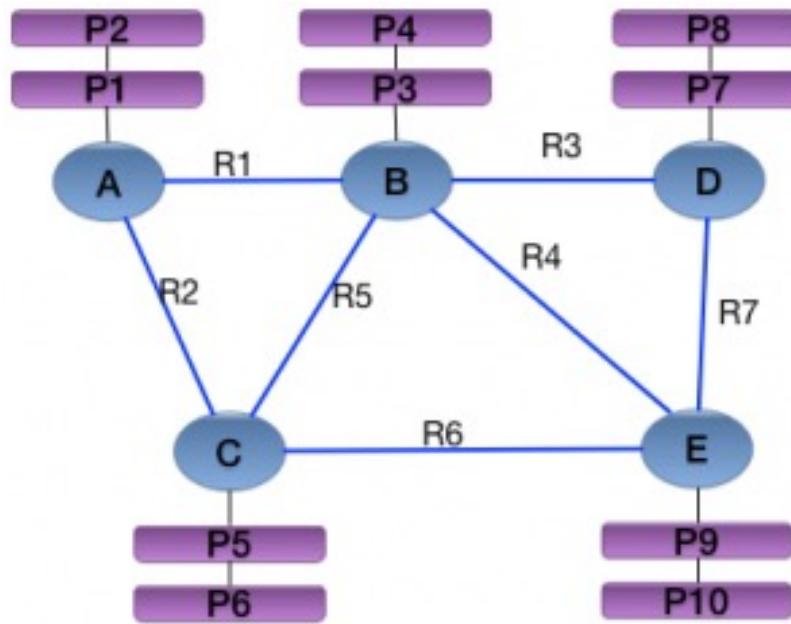
Graph Database Internals

- Neo4j stores graph data



Graph Database Internals

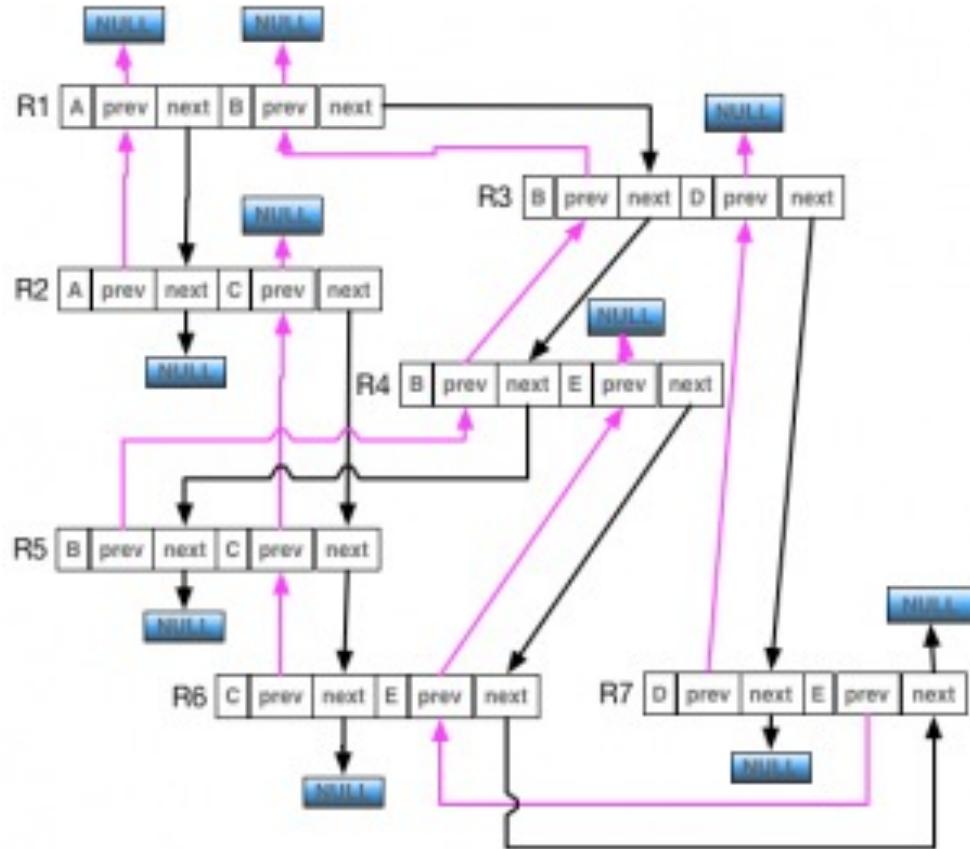
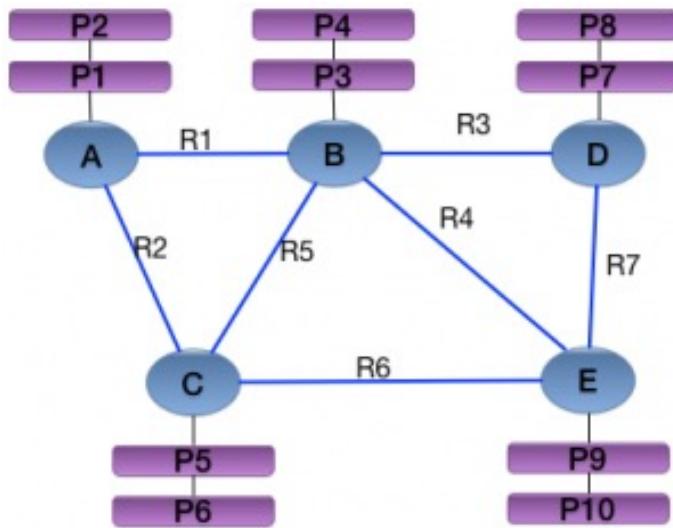
- Neo4j stores graph data
 - Node

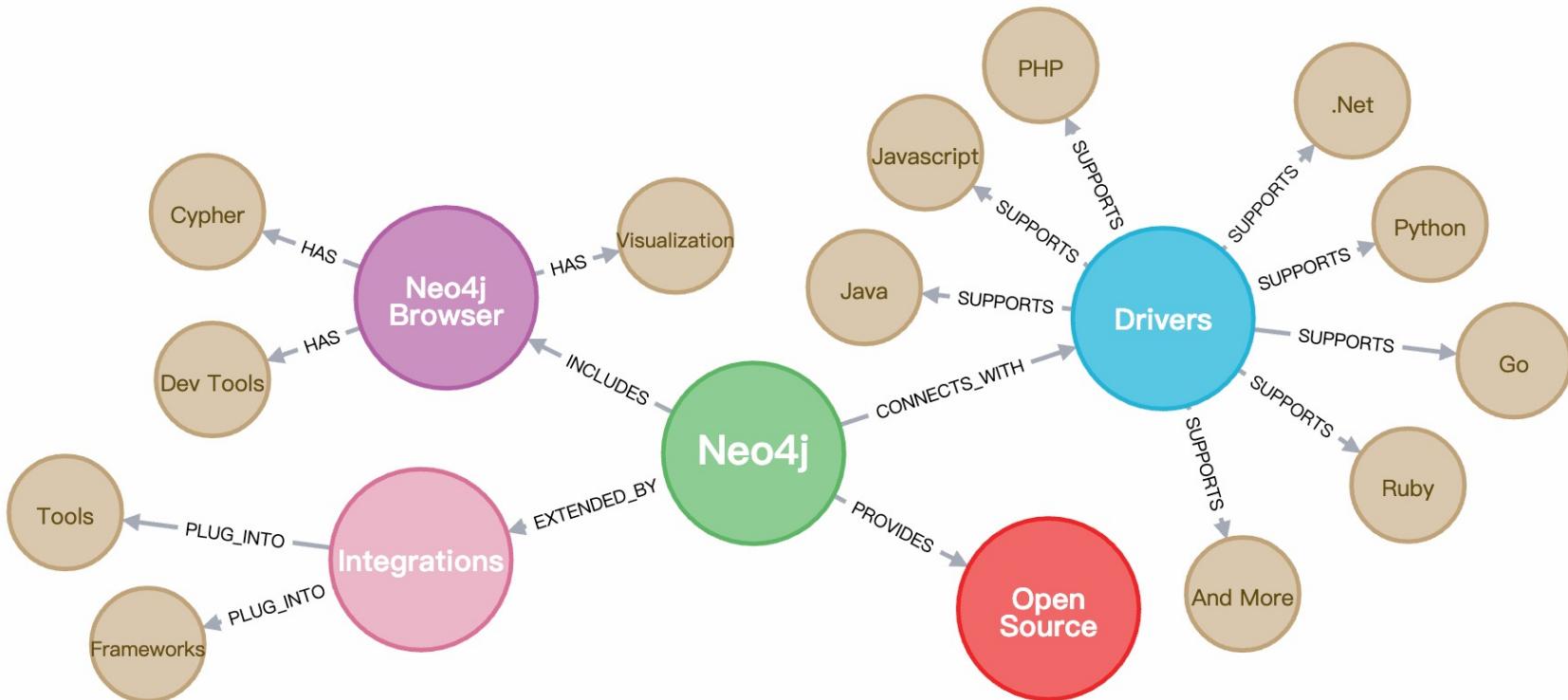


	inUse	nextRelId	nextPropId
	1	5	9
A:	1	R1	P1
B:	1	R1	P3
C:	1	R2	P5
D:	1	R3	P7
E:	1	R4	P9

Graph Database Internals

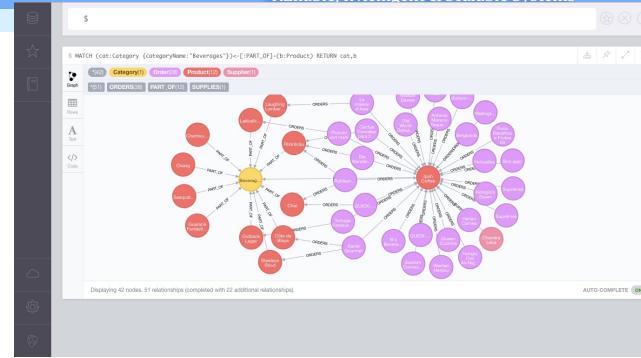
- Neo4j stores graph data
 - relationship





Install and Run Neo4j

1. Open up your terminal/shell.
2. Extract the contents of the archive, using:
`tar -xf <filecode>`.
For example,
`tar -xf neo4j-community-4.3.3-unix.tar.gz`
3. Place the extracted files in a permanent home on your server. The top level directory is referred to as NEO4J_HOME.
 - To run Neo4j as a console application, use:
`<NEO4J_HOME>/bin/neo4j console`
 - To run Neo4j in a background process, use:
`<NEO4J_HOME>/bin/neo4j start`
4. Visit <http://localhost:7474> in your web browser.
5. Connect using the username 'neo4j' with default password 'neo4j'. You'll then be prompted to change the password.



- pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-neo4j</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
        <exclusions>
            <exclusion>
                <groupId>org.junit.vintage</groupId>
                <artifactId>junit-vintage-engine</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
</dependencies>
```

- Person.java

```
@NodeEntity
public class Person {

    @Id @GeneratedValue
    private Long id;
    private String name;

    private Person() {
        // Empty constructor required as of Neo4j API 2.0.5
    };
    public Person(String name) {
        this.name = name;
    }

    @Relationship(type = "TEAMMATE", direction = Relationship.UNDIRECTED)
    public Set<Person> teammates;

    public void worksWith(Person person) {
        if (teammates == null) {
            teammates = new HashSet<>();
        }
        teammates.add(person);
    }
}
```

- Person.java

```
public String toString() {  
  
    return this.name + "'s teammates => "  
        + Optional.ofNullable(this.teammates).orElse(  
            Collections.emptySet()).stream()  
            .map(Person::getName)  
            .collect(Collectors.toList());  
}  
  
public String getName() {  
    return name;  
}  
  
public void setName(String name) {  
    this.name = name;  
}  
}
```

- PersonRepository.java

```
package com.example.accessingdataneo4j;

import java.util.List;
import org.springframework.data.neo4j.repository.Neo4jRepository;

public interface PersonRepository extends Neo4jRepository<Person, Long> {

    Person findByName(String name);
    List<Person> findByTeammatesName(String name);
}
```

- AccessingDataNeo4jApplication.java

```
@SpringBootApplication
@EnableNeo4jRepositories
public class AccessingDataNeo4jApplication {

    private final static Logger log = LoggerFactory.getLogger(AccessingDataNeo4jApplication.class);

    public static void main(String[] args) throws Exception {
        SpringApplication.run(AccessingDataNeo4jApplication.class, args);
        System.exit(0);
    }

    @Bean
    CommandLineRunner demo(PersonRepository personRepository) {
        return args -> {
            personRepository.deleteAll();

            Person greg = new Person("Greg");
            Person roy = new Person("Roy");
            Person craig = new Person("Craig");

            List<Person> team = Arrays.asList(greg, roy, craig);
        };
    }
}
```

- AccessingDataNeo4jApplication.java

```
List<Person> team = Arrays.asList(greg, roy, craig);

log.info("Before linking up with Neo4j...");

team.stream().forEach(person -> log.info("\t" + person.toString()));

personRepository.save(greg);
personRepository.save(roy);
personRepository.save(craig);

greg = personRepository.findByName(greg.getName());
greg.worksWith(roy);
greg.worksWith(craig);
personRepository.save(greg);

roy = personRepository.findByName(roy.getName());
roy.worksWith(craig);
// We already know that roy works with greg
personRepository.save(roy);
```

- AccessingDataNeo4jApplication.java

```
// We already know craig works with roy and greg

log.info("Lookup each person by name...");
team.stream().forEach(person -> log.info(
    "\t" + personRepository.findByName(person.getName()).toString()));

List<Person> teammates = personRepository.findByTeammatesName(greg.getName());
log.info("The following have Greg as a teammate...");
teammates.stream().forEach(person -> log.info("\t" + person.getName()));
};

}

}
```

Sample

SE343_16_neo4jsample [~/ideaProjects/SE343_16_neo4jsample] - .../src/main/java/org/reins/neo4jsample/Neo4jsampleApplication.java

Project: SE343_16_neo4jsample

File: pom.xml

```
package org.reins.neo4jsample;
import java.util.Arrays;
import java.util.List;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.data.neo4j.repository.config.EnableNeo4jRepositories;

@SpringBootApplication
@EnableNeo4jRepositories
public class Neo4jsampleApplication {

    private final static Logger log = LoggerFactory.getLogger(Neo4jsampleApplication.class);

    public static void main(String[] args) throws Exception {
        SpringApplication.run(Neo4jsampleApplication.class, args);
    }
}
```

Run: Neo4jsampleApplication

Console

```
2020-03-25 18:28:11.048 INFO 14372 --- [           main] org.neo4j.ogm.metadata.DomainInfo      : Starting Post-processing phase
2020-03-25 18:28:11.048 INFO 14372 --- [           main] org.neo4j.ogm.metadata.DomainInfo      : Building byLabel lookup maps
2020-03-25 18:28:11.053 INFO 14372 --- [           main] org.neo4j.ogm.metadata.DomainInfo      : Building interface class map for 3 classes
2020-03-25 18:28:11.106 INFO 14372 --- [           main] org.neo4j.ogm.metadata.DomainInfo      : Post-processing complete
2020-03-25 18:28:11.239 INFO 14372 --- [           main] o.s.d.neo4j.mapping.Neo4jMappingContext : Neo4jMappingContext initialisation completed
2020-03-25 18:28:11.296 INFO 14372 --- [           main] o.r.neo4jsample.Neo4jsampleApplication : Started Neo4jsampleApplication in 1.148 seconds (JVM running for 1.844)
2020-03-25 18:28:11.296 INFO 14372 --- [           main] o.r.neo4jsample.Neo4jsampleApplication : Direct driver instance 180191949035 created for server address localhost:7687
2020-03-25 18:28:11.551 INFO 14372 --- [           main] o.r.neo4jsample.Neo4jsampleApplication : Before linking up with Neo4j...
2020-03-25 18:28:11.551 INFO 14372 --- [           main] o.r.neo4jsample.Neo4jsampleApplication : Greg's teammates == []
2020-03-25 18:28:11.552 INFO 14372 --- [           main] o.r.neo4jsample.Neo4jsampleApplication : Roy's teammates == []
2020-03-25 18:28:11.552 INFO 14372 --- [           main] o.r.neo4jsample.Neo4jsampleApplication : Craig's teammates == []
2020-03-25 18:28:11.552 INFO 14372 --- [           main] o.r.neo4jsample.Neo4jsampleApplication : Lookup each person by name...
2020-03-25 18:28:12.184 INFO 14372 --- [           main] o.r.neo4jsample.Neo4jsampleApplication : Greg's teammates == [Roy, Craig]
2020-03-25 18:28:12.191 INFO 14372 --- [           main] o.r.neo4jsample.Neo4jsampleApplication : Roy's teammates == [Greg, Craig]
2020-03-25 18:28:12.198 INFO 14372 --- [           main] o.r.neo4jsample.Neo4jsampleApplication : Craig's teammates == [Roy, Greg]
```

Event Log

Build completed successfully in 3 s 467 ms (today 6:28下午)

neo4j@bolt://localhost:7687/neo4j - Neo4j Browser

neo4j\$

neo4j\$ MATCH (n:Person) RETURN n LIMIT 25

Graph Table Text Code

*3 Person(3)
*(4) TEAMMATE(4)

Craig Greg Roy

TEAMMATE TEAMMATE TEAMMATE

Displaying 3 nodes, 4 relationships.

\$:play start

neo4j

Learn about Neo4j
A graph epiphany awaits you.

Jump into code
Use Cypher, the graph query

System information
Key system health and status

The screenshot displays the Neo4j Browser interface. On the left, there's a sidebar with various configuration options: 'Database Information' (using neo4j - default), 'Node Labels' (Person), 'Relationship Types' (TEAMMATE), 'Property Keys' (name), 'Connected as' (Username: neo4j, Roles: admin, Admin: server user list, Admin: server user add), and 'DBMS' (Version: 4.0.1, Edition: Enterprise, Name: neo4j, Databases: .dbs, Information: .sysinfo, Query List: .queries). The main panel shows the results of a Cypher query: 'MATCH (n:Person) RETURN n LIMIT 25'. It lists three Person nodes (Craig, Greg, Roy) and four TEAMMATE relationships between them. Below the results, there are links to learn about Neo4j, jump into code, and view system information. At the bottom, there's a logo for neo4j and a message: 'Learn about Neo4j A graph epiphany awaits you.'

➤ 图数据上的分类问题

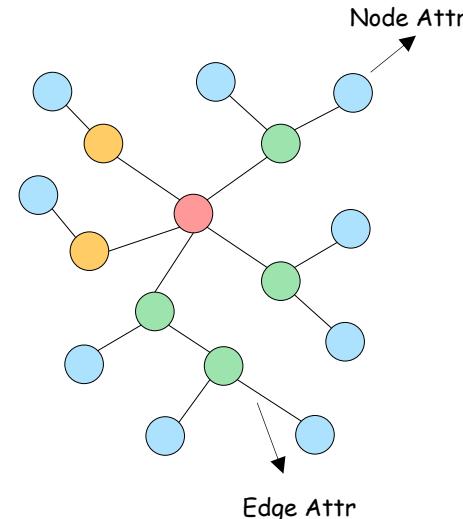
- 图粒度 vs. 节点粒度
- 节点特征 vs. 边特征

➤ 应用场景：

- 应用对象：社交网络、交易网络（边特征敏感）
- 应用任务：用户分类、异常行为检测…

➤ 图神经网络：

- 传统图神经网络未考虑边特征对于分类结果的影响
 - 谱图理论：Graph Convolutional Network (GCN)
 - 图注意力：Graph Attention Network (GAT)
- 仅有少数研究考虑了GNN在边特征上的拓展
 - 仅可处理标签化的边特征 (R-GCN)
 - 仅将边特征视为权重进行处理 (EGNN)



- 边特征的集合形式 → 邻接形式

$$\vec{e}_p \rightarrow \vec{e}_{ij}$$

- 边映射矩阵 \mathbf{M}_E

- 维度大小为 $N \times N \times M$
- 第三维度：one-hot编码
- 可以预先构造

- 高维矩阵乘法

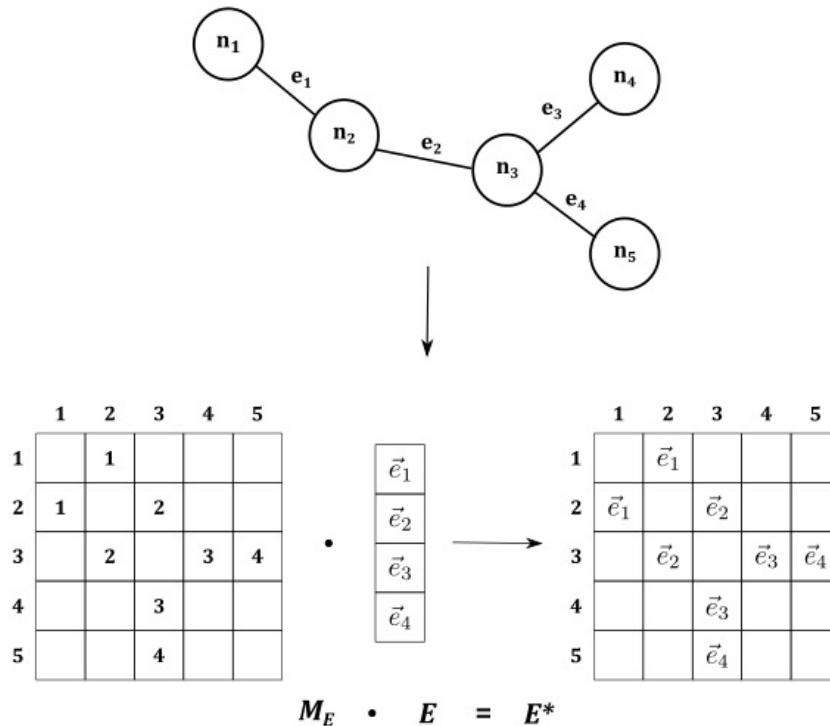


图 3-3 图上的边特征映射变换举例

节点注意力模块

- 节点特征的聚合（参与后续迭代）

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \vec{h}_j\right)$$

$$\sum_{j \in \mathcal{N}_i} \alpha_{ij} = 1$$

- 带有边特征的聚合（仅于融合层使用）

$$\vec{m}_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} (\vec{h}_j \| \vec{e}_{ij})\right)$$

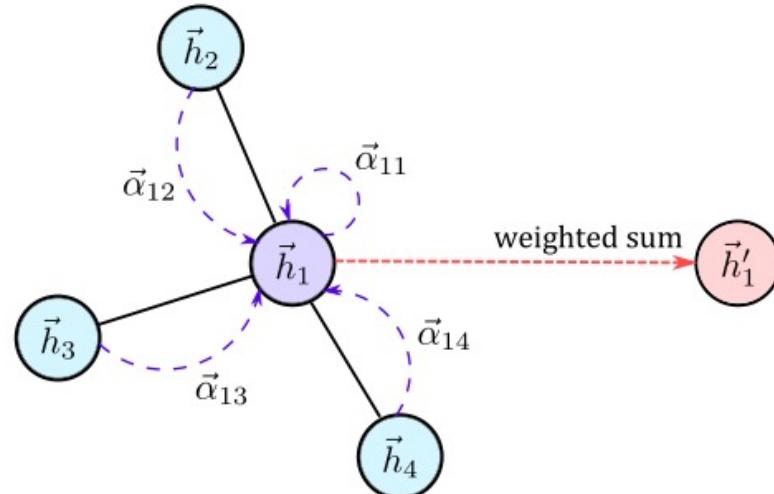


图 3-5 节点注意力模块中的节点特征聚合举例

注意力权重计算

- 带有边特征的注意力权重计算

$$w_{ij} = a(\vec{h}_i, \vec{h}_j, \vec{e}_{ij})$$

- 权重标准化 (softmax)

$$\alpha_{ij} = \text{softmax}_j(w_{ij}) = \frac{\exp(w_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(w_{ik})}$$

- 展开

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\vec{a}^T [\vec{h}_i \| \vec{h}_j \| \vec{e}_{ij}]))}{\sum_{k \in \mathcal{N}_i} \exp(\text{LeakyReLU}(\vec{a}^T [\vec{h}_i \| \vec{h}_k \| \vec{e}_{ik}]))}$$

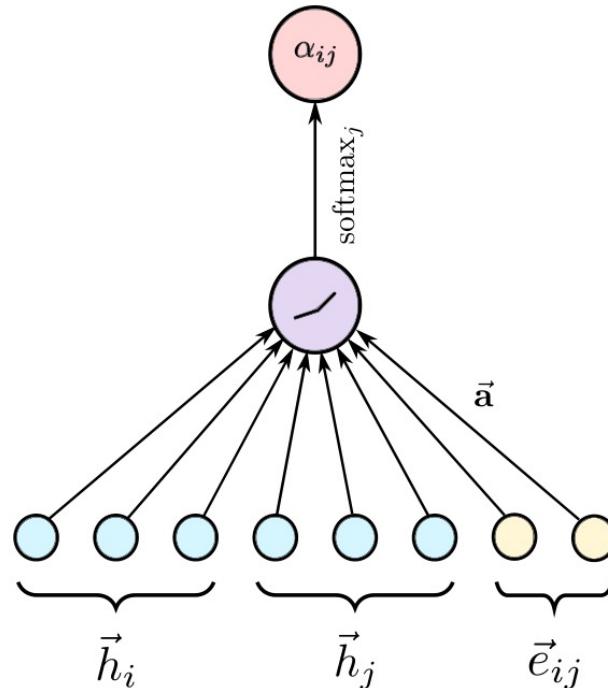


图 3-4 标准化注意力权重计算方式举例

➤ 边特征迭代的必要性

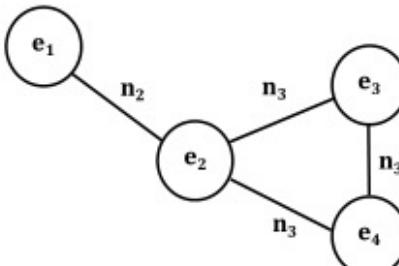
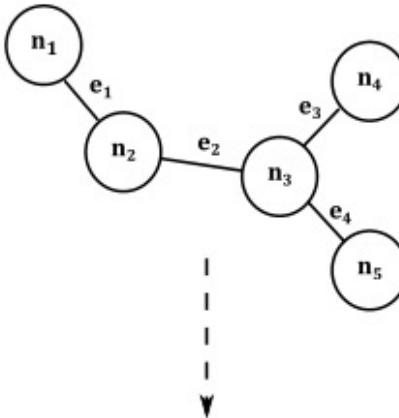
- 节点与边特征在迭代上的一致性
- 模型的对称性

➤ 点边互换策略

- 反转图
- 边邻接矩阵、节点映射矩阵

$$\beta_{pq} = \frac{\exp(\text{LeakyReLU}(\vec{\mathbf{b}}^T [\vec{e}_p \| \vec{e}_q \| \vec{h}_{pq}]))}{\sum_{k \in \mathcal{N}_p} \exp(\text{LeakyReLU}(\vec{\mathbf{b}}^T [\vec{e}_p \| \vec{e}_k \| \vec{h}_{pk}]))}$$

$$\vec{e}'_p = \sigma \left(\sum_{q \in \mathcal{N}_p} \beta_{pq} \vec{e}_q \right)$$



a) graph transformation

	1	2	3	4
1	1	1		
2	1	1	1	1
3		1	1	1
4		1	1	1

b) edge adjacency matrix

	1	2	3	4
1		2		
2	2		3	3
3		3		3
4		3	3	

c) node mapping matrix

图 3-7 边特征迭代相关的图转换过程及相关矩阵图示

- 多尺度的特征融合策略

$$\vec{h}_i^* = \left\| \begin{array}{c} l \\ m_i^l \end{array} \right\|_{l=1}^L$$

- 粗粒度的multi-head attention

$$\vec{h}_i^* = \left\| \begin{array}{c} k \\ l \\ m_i^{l,k} \end{array} \right\|_{k=1}^K \left\| \begin{array}{c} l \\ m_i^l \end{array} \right\|_{l=1}^L.$$

- 维度压缩

$$\vec{h}_i^f = \text{LeakyReLU}(\mathbf{W}\vec{h}_i^*)$$

- softmax

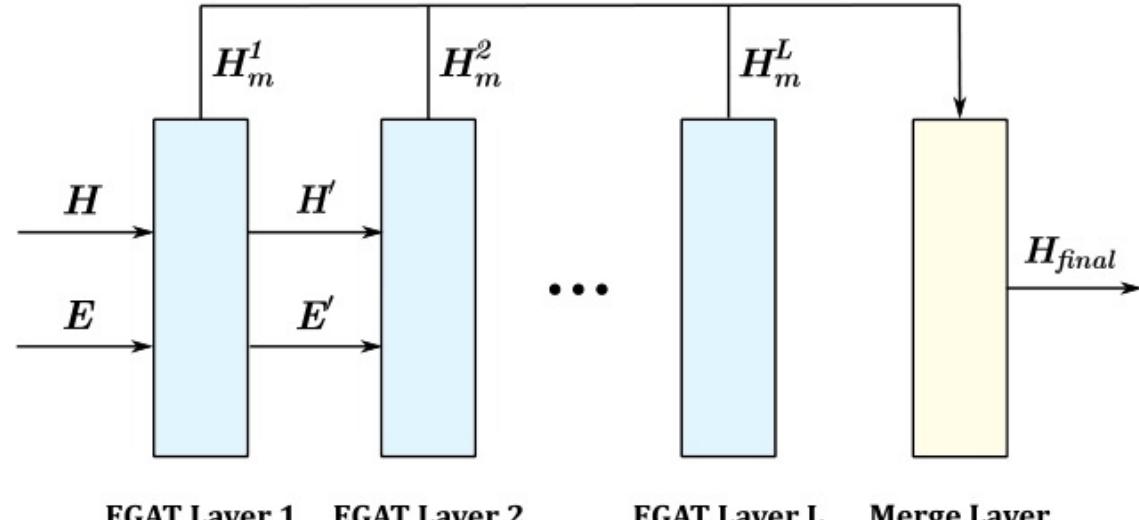


图 3-8 EGAT 融合层及与其相关的输入输出实例

稀疏矩阵表示

- 邻接矩阵 & 映射矩阵
 - 维度大小 ~ 图规模关系的非线性
 - 内存与显存的局限性 & 计算的复杂性
- 稀疏矩阵表示
 - 空元素数量 >> 非空元素数量
 - 高维映射矩阵的初次压缩

		0	1	2	3
0				1	
1			1		
2		1		1	
3	1		1		

0	1	2	2	3	3
3	2	1	3	0	2
1	1	1	1	1	1

图 4-1 COO 格式下稀疏矩阵表示形式举例

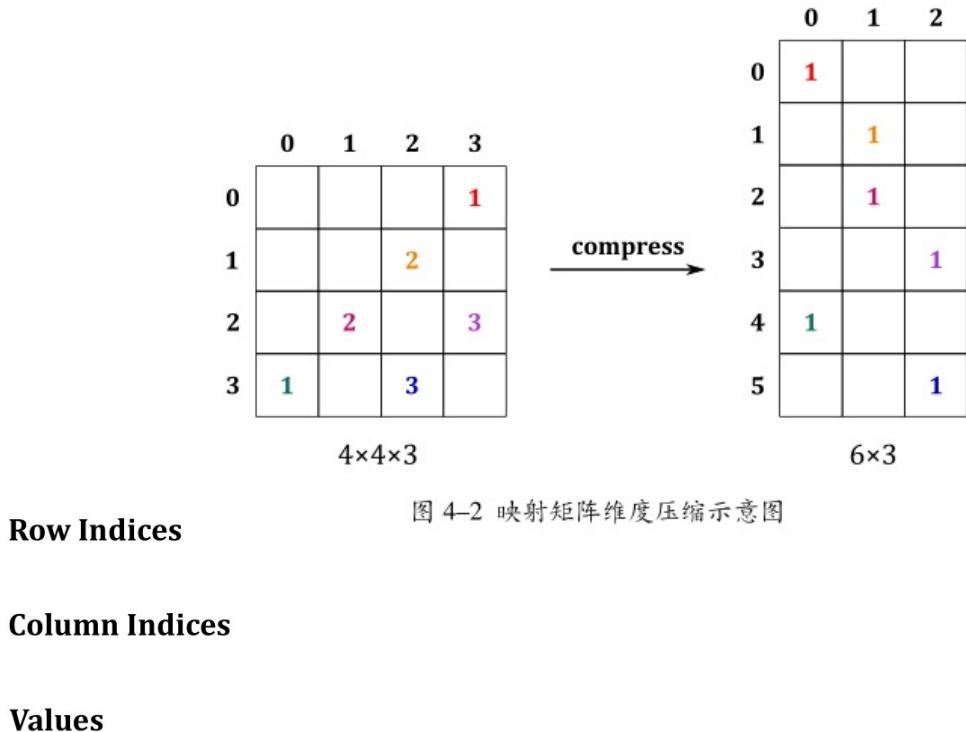
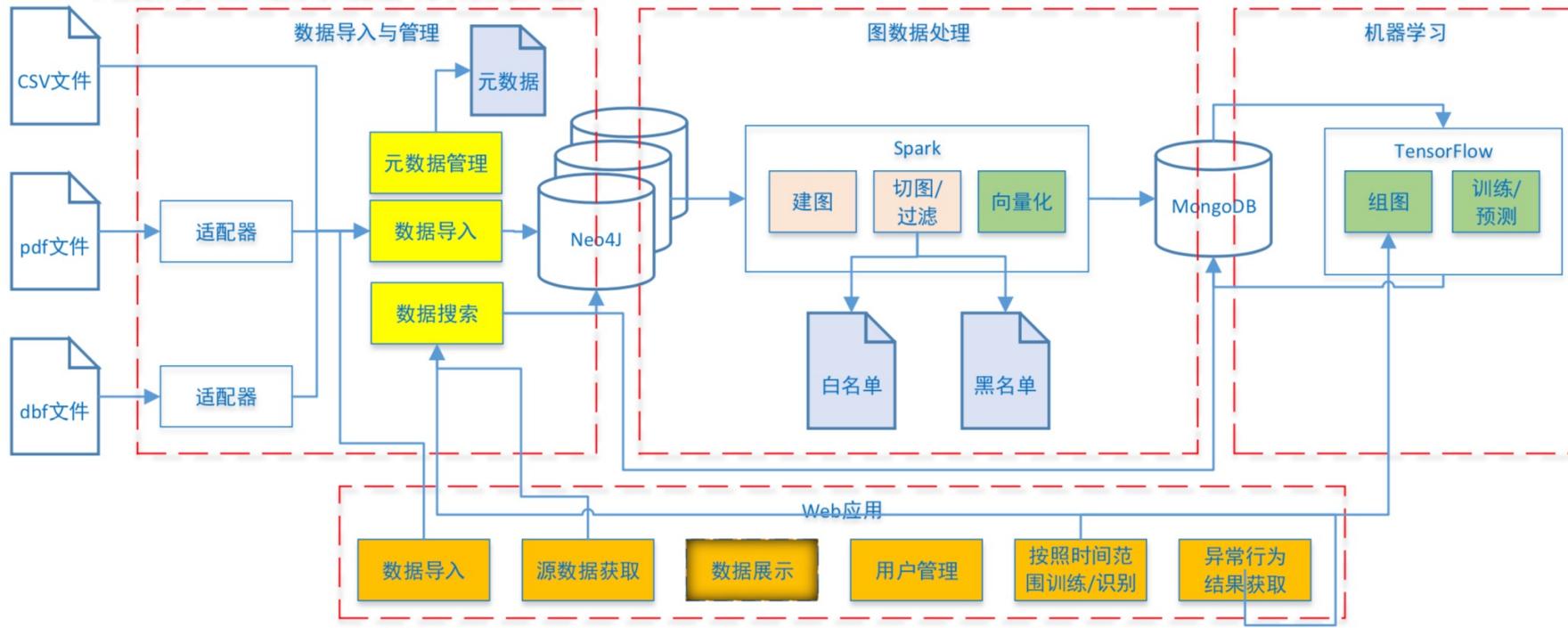


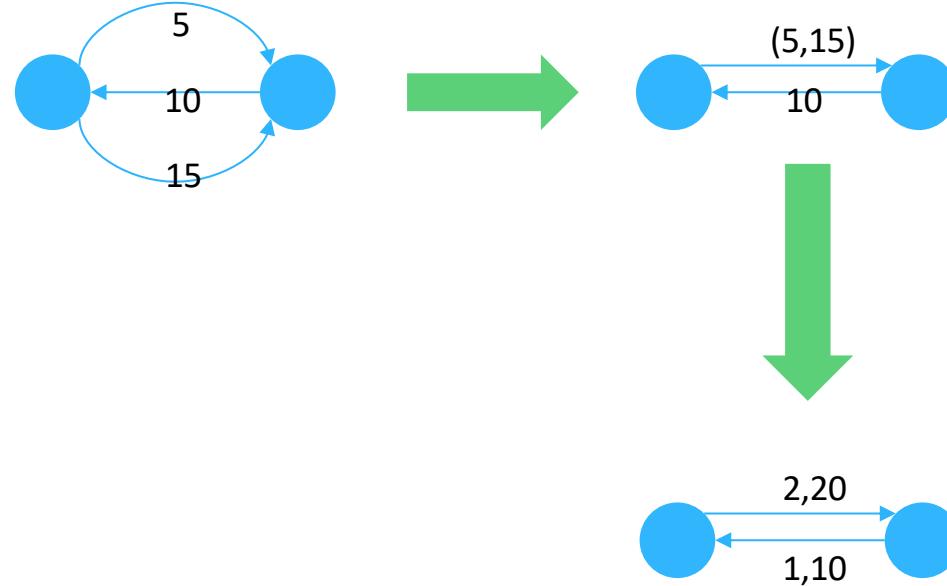
图 4-2 映射矩阵维度压缩示意图

总体流程与架构设计

- 根据实体关联数据构建图 -> 根据样例学习图中的模式 ->发现符合预期模式的数据，进行预警

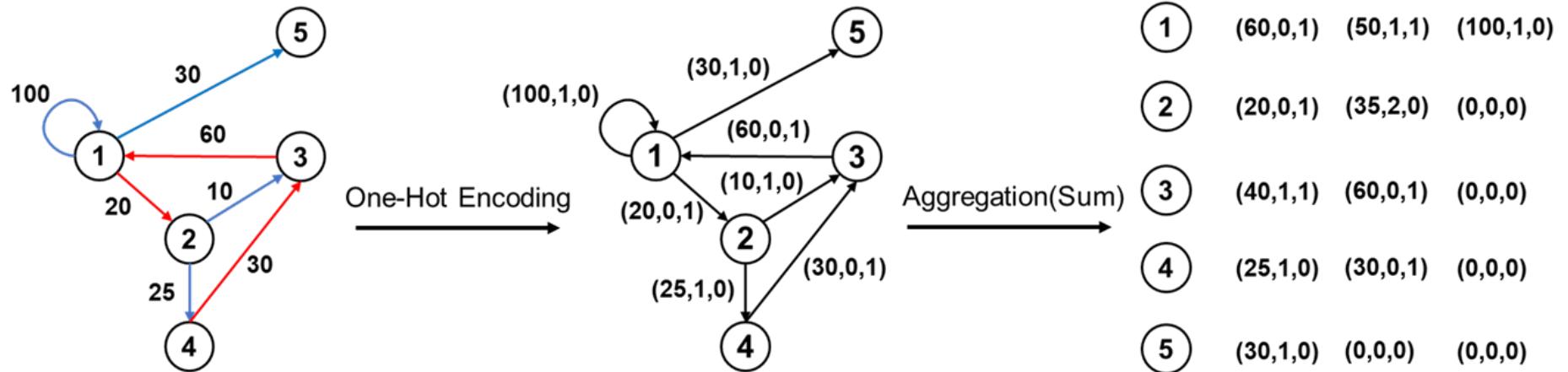


- Multi-Graph 多边合并



边属性合并

- 边属性 -> 节点属性

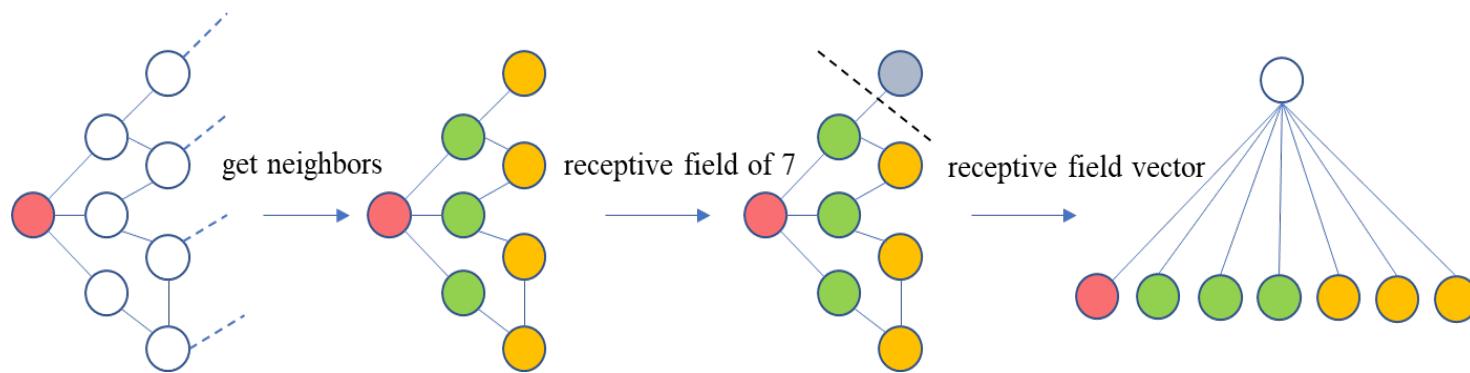


- Graph -> Vector

```
data = {  
    time: 2019-01, 时间  
    type: train, 数据类型  
    data: 数据  
    [  
        {  
            id: 1, 中心节点ID  
            label: 1, 中心节点标签  
            neighbors: [{ 邻居节点  
                id: 1, 邻居节点ID  
                attr: {  
                    layer: 0, 距中心节点距离  
                    origin: [1.0, 1.0],  
                    summable: [1.0, 2.0]  
                },  
                ...  
            }]  
        },  
        ...  
    ]  
}
```

邻居节点原始属性
邻居节点可加属性
(边合并属性)

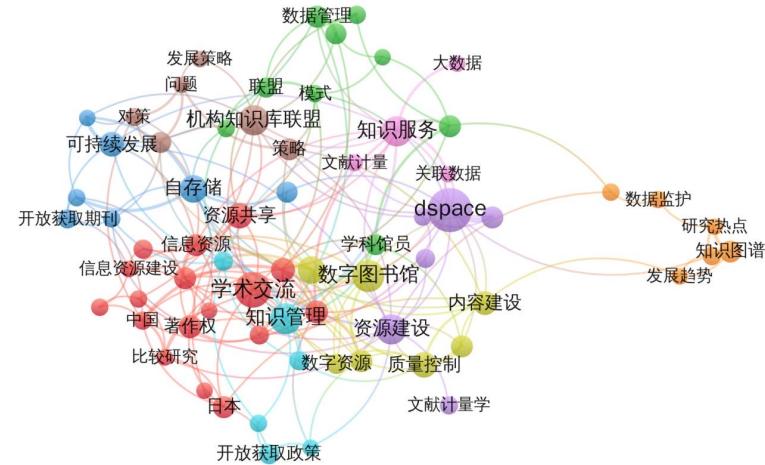
- GraphX (Pregel迭代计算)
- 设置传播层数上限
- 固定的邻居数量 (Fixed-Size)
- 结果存储到MongoDB中 (Hiding Processing Latency)



- 请你在大二开发的E-Book系统的基础上，完成下列任务：
 1. 将你认为合适的内容改造为在MongoDB中存储，例如书的产品评价或书评。你可以参照课程样例将数据分别存储在MySQL和MongoDB中，也可以将所有数据都存储在MongoDB中，如果采用后者，需要确保系统功能都能正常实现，包括书籍浏览、查询、下订单和管理库存等。
 - 请将你编写的相关代码整体压缩后上传，请勿压缩整个工程提交。
- 评分标准：
 1. 能够正确实现上述数据存储方案。(3分)

作业八(第二部分)

- 请你在大二开发的E-Book系统的基础上，完成下列任务：
 1. 为你的每一本图书都添加一些标签，在Neo4J中将这些标签构建成一张图，类似右图。
 2. 在系统中增加一项搜索功能，如果用户按照标签搜索，你可以将Neo4J中存储的与用户选中的标签以及通过2重关系可以关联到的所有标签都选出，作为搜索的依据，在MySQL中搜索所有带有这些标签中任意一个或多个的图书，作为图书搜索结果呈现给用户。
 - 请将你编写的相关代码整体压缩后上传，请勿压缩整个工程提交。



- 评分标准：
 1. 能够正确实现图书标签图在Neo4J中的构建与存储，并实现针对标签图的模糊搜索功能(2分)

- Graph Databases: NEW OPPORTUNITIES FOR CONNECTED DATA
 - <http://graphdatabases.com/?ref=blog& ga=2.114598147.542116335.1524369358-1583480045.1524187154>
- Neo4j图数据库简介和底层原理
 - <https://blog.csdn.net/bluejoe2000/article/details/72978478>
- Neo4j install
 - <https://neo4j.com/download-thanks/?edition=community&release=4.3.3&flavour=unix>
- Accessing Data with Neo4j
 - <https://spring.io/guides/gs/accessing-data-neo4j/>



Thank You!