Московский ордена Ленина, ордена Октябрьской Революции и ордена Трудового Красного Знамени государственный технический университет им. Н.Э. Баумана

Реферат
по курсу «Постреляционные базы данных»
кафедры ИУ-5
«Графовая СУБД NEO4»

Проверила:
Виноградова М.В.
Выполнила:
Студент группы ИУ7-39М
_____/ Щукина В.В

Москва 2017 г.

Содержание

Терминология Neo4j и графовых баз данных в целом.

- **graph database, графовая база данных** — база данных построенная на графах — узлах и связях между ними
- **Cypher** — язык для написания запросов к базе данных Neo4j (примерно, как SQL в MYSQL)
- **node, нода** — объект в базе данных, узел графа. Количество узлов ограниченно 2 в степени 35 ~ 34 биллиона
- **node label, метка ноды** — используется как условный «тип ноды». Например, ноды типа movie могут быть связанны с нодами типа actor. Метки нод — регистрозависимые, причем Cypher не выдает ошибок, если набрать не в том регистре название.
- **relation, связь** — связь между двумя нодами, ребро графа. Количество связей ограниченно 2 в степени 35 ~ 34 биллиона
- **relation identirfier, тип связи** — в Neo4j у связей. Максимальное количество типов связей 32767
- **properties, свойства ноды** — набор данных, которые можно назначить ноде. Например, если нода — это товар, то в свойствах ноды можно хранить id товара из базы MySQL
- **node ID, ID нода** — уникальный идентификатор ноды. По умолчанию, при просмотрах результата отображается именно этот ID. как его использовать в Cypher запросах я не нашел

Графовая база данных

Графовая база данных — это такая база данных, которая использует графовые структуры для построения семантических запросов с использованием узлов, ребер и свойств в процессе представления и хранения данных. Графовые БД используются для хранения, управления и составления запросов к сложным и тесно взаимосвязанным группам данных. Помимо этого, архитектура графовой БД особенно хорошо приспособлена к анализу данных на предмет обнаружения совпадений и аномалий в обширных массивах данных, а также к выгодному использованию заключенных в БД взаимосвязей.

Графовые модели баз данных появились наряду с объектно-ориентированными моделями. Эти модели пытаются преодолеть ограничения, накладываемые традиционными моделями. Мотивацией для разработки таких систем стала графовая структура данных, появляющихся в таких приложениях, как гипертекстовые или географические информационные системы, где взаимосвязь между данными является важным аспектом. Полуструктурированными моделями называют такие концепции, которые предназначены для хранения данных с гибкой структурой, например, документов и веб-страниц [1].

Формально, граф является набором вершин и ребер. Графы представляют собой связи как узлы и как способы, которыми эти объекты связаны друг с другом в качестве отношений. В общем смысле, такая структура позволяет моделировать все виды сценариев, от строительства космической ракеты до системы дорог. Графы полезны для понимания широкого разнообразия наборов данных в таких областях как наука, управление или бизнес [2]. В дальнейшем, в концепцию графовой модели базы данных будут включены 9 три основных компонента, а именно:

- структура данных, язык преобразования данных и ограничение их целостности. Следовательно, графовые модели базы данных характеризуется следующим образом [2]:

данные и/или схемы представлены в виде графов, или с помощью структур данных, которые в итоге сводятся к графам (гиперграфы или гипервершины);

- манипуляция над данными выражается либо в виде преобразовании графов, либо операциями, примитивы которых являются графовые структуры, такие как пути, подграфы, формы графа, связи и прочие;
- согласование данных обеспечивается с помощью ограничений целостности. Эти ограничения могут быть сгруппированы в последовательности схемы, например, идентичности и ссылочной целостности, и функциональные зависимостей включения [2].

Компоненты графовой базы данных — узлы и ребра. Они могут быть дополнены собственным набором полей. Модель такой БД схематично изображена на рисунке 1.

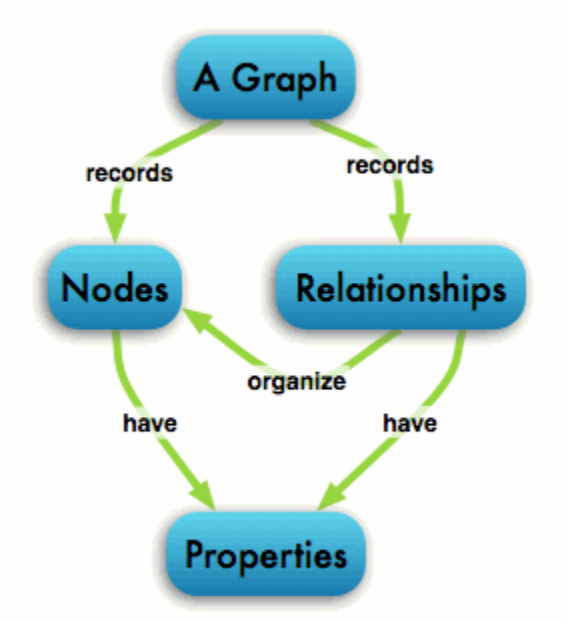

Рисунок 1. Модель графовой базы данных

Графовая СУБД Neo4j

Neo4j — open-source графовая база данных, история которой началась по инвестициям компании «Neo Technology» в 2003 году. С 2007 года стала публично доступной. В Neo4j присутствуют все характеристики баз данных, включая соблюдение ACID, поддержание разбиение на кластеры и восстановления после сбоя в системе. Сегодня является лидером среди графов баз данных.

ACID (Atomicity, Consistency, Isolation, Durability) — набор свойств, которые гарантируют надежную работу транзакций.

Neo4j позволяет быстро разрабатывать системы, основанные на графовой модели данных. Архитектура Neo4j разработана для оптимизации таких процессов а базе, как управление, хранение и обход узлов и связей. В Neo4j, отношения являются важными объектам, которые представляют собой связи между сущностями. Операция, известная в реляционной базе как соединение (JOIN), производительность которой падает экспоненциально с числом отношений, в Neo4j осуществляется как навигация от одного узла к другому, алгоритм работы которого является линейным [7].

Этот другой подход к хранению и запросам связей между объектами обеспечивает скорость выполнения обхода графа до 4 миллионов вершин в секунду, в зависимости от параметра вычислительной системы. Поскольку большинство задач по поиску в графе чаще всего затрагивает локальные окрестности вершины, общий объем данных, который сохранен в базе данных, не влияет на общую скорость выполнения операций.

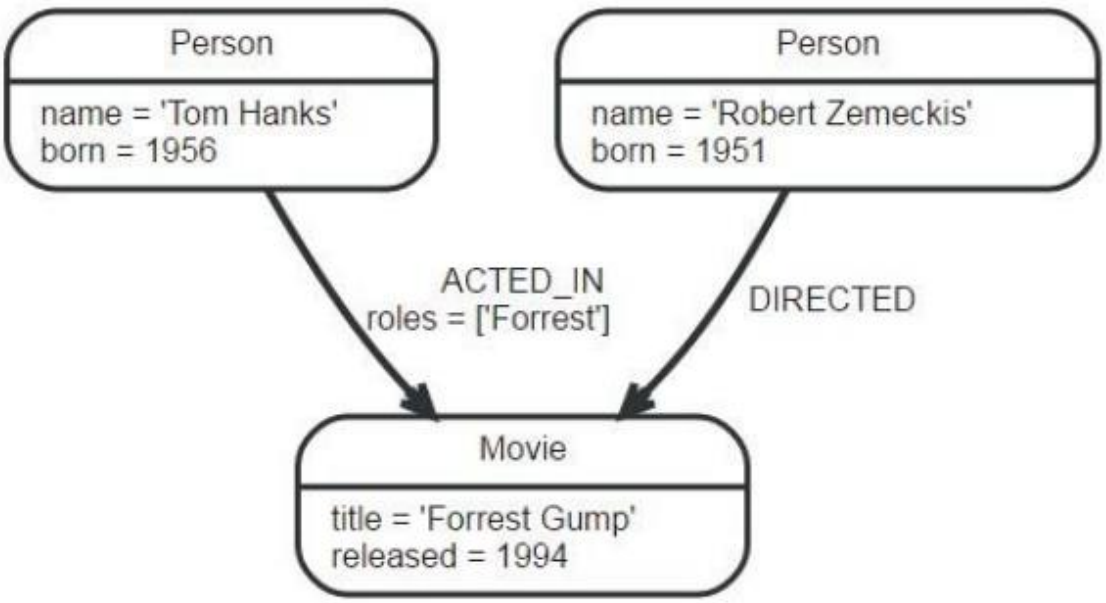

Рисунок 2. Представление данных в СУБД Neo4j

Графовая СУБД хранит данные в графе, которая, являясь довольно гибкой структурой данных, способна изящно представить любые данные в высоко доступной форме [7].

Далее опишем объекты, которые предоставляет Neo4j:

- вершины (nodes) - используются для представления сущностей, но в зависимости от отношений в графе могут быть использованы для представления связи. Самый простой граф представляет собой одну вершину. Вершина может не иметь значить, либо иметь одно или более именованных значений, которые указываются в виде свойств;
- связи (relationships) – организовывают узлы, соединяя их. Связь соединяет два узла – начальный и конечный. Так же, как и узлы, связи могут иметь свойства;
- свойства (prosperties) – именованные значения, где имя – это строка. Поддерживаемые значения: числовые, строковые, двоичные, списки предыдущих типов;
- метки (labels) - предоставляют собой графы, которые были сгруппированы в наборы. Все узлы, помеченные одной меткой, принадлежит к одному набору. Упрощают написание запросов к базе. Вершина может быть помечена любым 20 количеством меток. Метки используются для задания ограничений и добавления индексов для свойств.

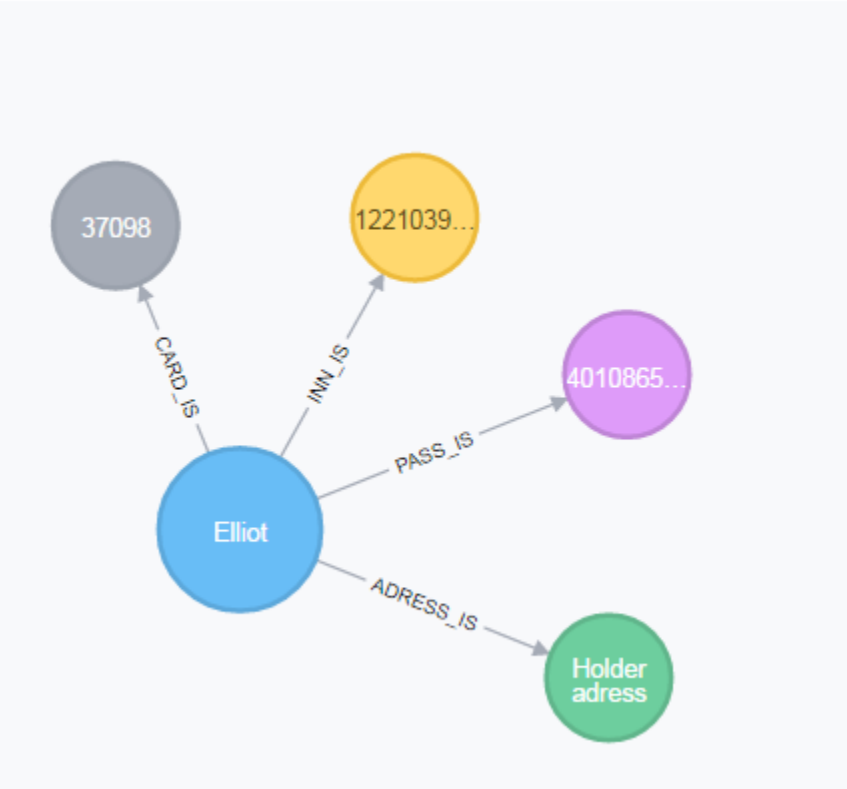

Рисунок 3. Графическое изображение данных в СУБД Neo4j

Производительность в Neo4j, как и в реляционных СУБД, достигается за счет создания индексов, которые увеличивают скорость поиска узлов в базе данных. После того, как свойства индекса были определены, Neo4j будет поддерживать индексы в актуальном состоянии, вместе с добавлением новых данных в базу. Любая операция, основанная на индексированных данных, будет иметь значительный рост производительности. В СУБД Neo4j также реализован графический интерфейс, на котором удобно представлены графовые данные. Пример изображения графа и данные, которые связаны с ним, представлены на рисунке 2.3. В дальнейшем некоторые результаты запросов будут представлены в таком же виде. Neo4j также поддерживает ограничения, которые можно наложить на данные. Ограничения позволяют определить правила того, какой вид должна иметь 21 информация. Любые изменения, которые нарушают эти правила, не будет применены.

Сохранение данных в Neo4j

Файл nodestore.db содержит определенный размер записей, содержащих информацию о Ноды:
1. Метка, которая показывает, запись активна;
2. Указатель на первое отношение, которое содержит данная нода;
3. Указатель на первую свойство, которое содержит данная нода.

Нода не содержит собственного идентификатора. Так как каждая запись в nodestore.db занимает одинаковое количество места, можно рассчитать указатель на ноду.

Файл relationshipstore.db также содержит записи одинакового размера, которые описывают отношения, но они состоят из следующих элементов:

1. Метка, которая показывает, запись активна;
2. Указатель на ноду, которая содержит это отношение;
3. Указатель на ноду, к которой это отношение направлено;
4. Вид отношения;
5. Указатель на отношение, которое стоит впереди (в пределах данной ноды);
6. Указатель на отношение, которое стоит сзади (в пределах данной ноды);
7. Указатель на отношение, которое стоит впереди (в пределах Ноды, в которой это отношение направлено);
8. Указатель на отношение, которое стоит сзади (в пределах Ноды, в которой это отношение направлено);
9. Указатель на первое свойство данного отношения.

Язык запросов Cypher

Будучи декларативным языком, Cypher фокусируется на ясности выражения того, что необходимо извлечь из графа, а не в том, как получить это. Это отличает его от императивных языков или скриптовых сценариев. Такой

подход существенно упрощает процесс оптимизации, не обременяя пользователя информацией о структуре базы данных и не вынуждая обновлять код запроса только потому, что логическая структура базы данных изменилась (новые индексы и т.д.) [8] .

Примеры обозначения искомых данных:

- (n)-->(m) — все направленные ребра из вершины n в вершину m;
- (n:Holder) — все вершины с меткой Holder;
- (n:Holder:Account) — все вершины, имеющие обе метки Holder и Account;
- (n:Holder {name:{Mark}}) — все вершины с меткой Holder и отфильтрованные по дополнительному свойству name;
- (n:Holder)-->(m) — ребра между вершинами n с меткой Person и m;
- ()-[:IS_HOLDER]->() — отобразить вершины, которые связаны между собой именованной связью;
- p = (n:Holder)-[*3..5]-(m:Holder) – отобразить все вершины, которые соединены между собой минимум через 3 другие вершины и максимум через 5, игнорируя направления связей.

Cypher поддерживает запросы с параметрами. Это означает, что разработчики не должны прибегать к строковому построению. Более того, параметры облегчают операцию кэширования исполнения планов для Cypher.

Параметры могут быть использованы для литералов и выражений в предложении WHERE, для значения индекса в предложении START, для индексированных запросов и для самих узлов или отношений. Параметры, не могут быть использованы в качестве имен свойств, типов отношений или меток.

Одно из главных преимуществ языка Cypher – выразительный SQL-подобный синтаксис, привычный для большинства пользователей баз данных. Кроме того, независимо от потенциального расширения своих возможностей этот язык будет оставаться достаточно простым [8].

Использование **Neo4j** на примере базы данных банка

База данных банка будет сформирована из двух сущностей:

- держатель (клиент) – хранит такую информацию о клиенте, как имя, фамилия, отчество, номер паспорта, пол и другие [5];
- кредитная карта – хранит номер карты, баланс и ее статус [5].

В графовой СУБД вся структура задается в виде отношений, без необходимости формирования в виде запросов. Далее опишем запросы, которые впоследствии будут выполняться в созданной информационной системе.

Запрос 1: Вывести фамилии всех клиентов, у которых отрицательный баланс на кредитной карте и чей номер паспорта имеет определенную последовательность цифр Этот запрос является достаточно простым, особенно для реляционных моделей баз, где нужно использовать оператор объединения таблиц. Такой запрос будет интерес тем, что можно будет посмотреть насколько эффективно графовая СУБД справляется с типовыми задачами.
Запрос 2: Для каждого имени держателя в базе найти сумму балансов всех кредитных карт и найти их количество. 16 Этот запрос является экспериментом на применение групповых функций. Так как язык Cypher также имеет набор инструментов для работы с группами, было решено изучить то, как они работают в действии.
Запрос 3: Отобразить таких клиентов в базе, которые могут быть любым способом связаны сами с собой через других клиентов. На выходе система должна отобразить путь, который можно проложить до этих данных, используя другие данные.

Разработка информационной системы на СУБД Neo4j

СУБД Neo4j хранит данные в гибком виде, в каком они представлены в реальном мире, используя для этого вершины и связи. Поэтому для графовой СУБД отсутствует такое понятие, как логическая модель данных, из чего следует, что стадия проектирования базы данных сильно упрощается.
Минимальной необходимостью являются договоренности в таких аспектах, как наименования связей, свойств и меток, которые будут использоваться для идентификации данных, принадлежащих к одному классу [8].

```
CREATE
(hld:Holder { id: 5, name: 'Karl', surname: 'Marx',
gender: 'M' }),
(adr:Adress { value: 'Sherman Avenue, 124' }),
(pass:Passport { value: '9813-166771' }),
(inn:Inn { value: '123-456-789-123' }),
(snils:Snils { value: '333-444-555-666' }),
(hld)-[:PASS_ID]->(pass),
(hld)-[:ADRESS_IS]->(adr)
(hld)-[:INN_IS]->(inn),
(hld)-[:SNILS_IS]->(snils);
```

Запрос представляет из себя команду CREATE,состоящую из двух частей. Первая - вида (…) - задает вершины графа и свойства этих вершин.
Вторая – вида (…)-[…]-(…) – задает отношения между вершинами, которые создаются в этом же запросе.
После импорта данных необходимо создать кольцо данных для теста запроса 3. Хотя бы один набор строк должен принять вид, который представлен рисунке 4. По данному графу видно, что четыре клиента имеют такие данные, которые позволяют провести транзитивную связь между вершинами, которые выражают держателей банковских профилей.

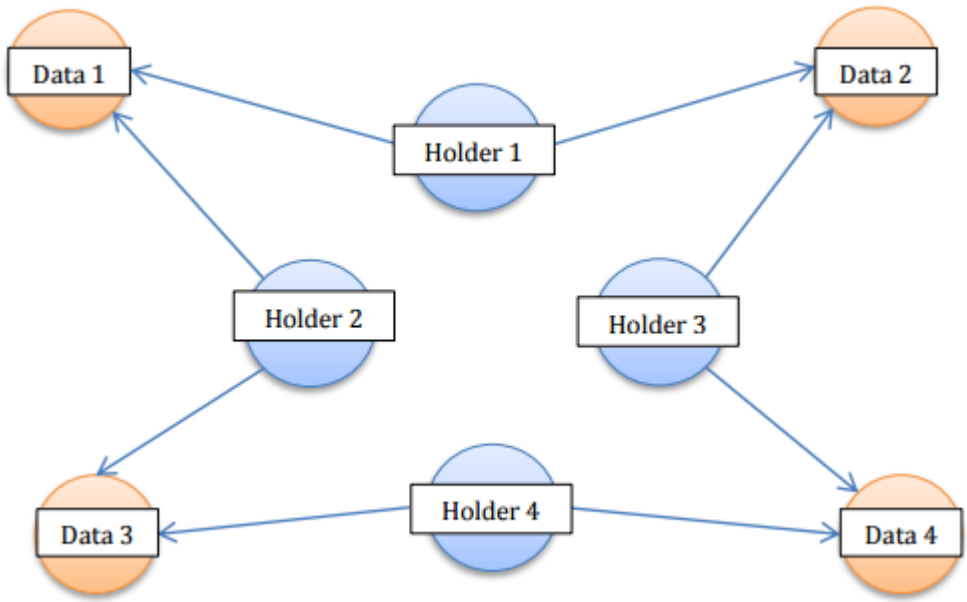

Рисунок 4. Граф, который образует кольцо

Показан вид запросов на языке Cypher, который позволяет задать граф, изображенный на рисунке 4:

```
MATCH
(hld1:Holder {id:5000})-[]-(inn1:Inn),
(hld2:Holder {id:400})-[]-(pass1:Passport)
MERGE (hld1)-[:PASS_IS]->(pass1)
….
MATCH
(hld2:Holder {id:400})-[]-(pass1:Passport),
(hld3:Holder {id:500})-[]-(inn2:Inn)
MERGE (hld2)-[:INN_IS]->(inn2);
```

С помощью команды MATCH база находит клиентов с определенным идентификатором, а также те данные, к которым задана связь в графовой СУБД, и задает для них определенные переменные. Используя команду MERGE и полученные переменные, происходит создание связей, в связи с поставленной задачей.

Запросы

Запрос 1.

Объединение таблиц Первый запросы был выбран с целью увидеть разницу в реализации запросов с использованием нескольких таблиц для реляционной СУБД. В связи с тем, что в СУБД Neo4j отсутствует такая логическая единица как таблица, получить данные из различных вершин можно используя оператор […], который обозначает связи. Регулярные выражения в запросах Cypher используют стандарты Java.

```
MATCH (hld: Holder)
-[:HOLD_CARD]-(card:   CreditCard),
(hld: Holder)-[:PASS_ID]-(pass: Passport)
WHERE card.balance < 0 AND pass.value =~ '.*192.*'
RETURN hld.name,
hld.surname, pass.value, card.balance
ORDER BY hld.name;
```

hld.name	hld.surname	pass.value	card.balance
Aaron	Parsons	4010591920	-12360
Aaron	Palmer	4010660192	-31385
Aaron	Lindsey	4010806192	-28634
Abram	Beasley	4010741929	-16656
Abram	Duncan	4010192402	-32651

Рисунок 5. Результат запроса 1.

Запрос 2. Агрегатные функции

В примере данного запроса используются операторы SUM() и COUNT(), которые схожи в обеих исследуемых СУБД. Первый оператор выполняет суммирование значений для указанного поля в базе и может применяться только для 30 числовых значений. Второй оператор выполняет суммирование общего количества полученных строк в пределах группы. Для СУБД Neo4j запрос будет выглядеть следующим образом:

```
MATCH (hld:Holder)-[:CARD_IS]-(card: CreditCard)
RETURN hld.name,  sum(card.balance), count(card)
ORDER By hld.name;
```

hld.name	sum(card.balance)	count(card)
Aaron	-1182073	2017
Abram	742323	2002
Alexzander	-380504	1919
Antoine	-2578103	2012
Augustus	-563778	1962

Рисунок 6. Результат запроса 2.

Запрос 3. Поиск связанных данных в графе

Используя особенности языка Cypher, запрос на поиск связанный данных в графе является простой задачей

```
MATCH
path=(hld:Holder)-[*]-(hld:Holder)
RETURN DISTINCT hld.name, hld.surname;
```

Язык Cypher является декларативным языком, который устроен так, чтобы запросы были понятны человеческому языку. Можно перевести этот запрос как “необходимо найти таких клиентов, до которых можно проложить путь к самим себе через граф, вернуть имена и фамилии клиентов”. Результат такого запроса можно увидеть ниже. Если изменить код после команды RETURN на nodes(path), то можно получить весь путь в виде графа, который

отображается в графическом виде в Neo4j. Результат такого запроса представлен на рисунке, где видно, как 6 клиентов разделяют такие общие данные, как номер телефона, домашний адрес и ИНН.

hld.name	hld.surname
Elliot	Fritz
Nasir	Torres
Nathaniel	Kerr
Gregory	Cowan
Yair	Parsons
Maxim	Duncan

Рисунок 7. Результат выполнения запроса.

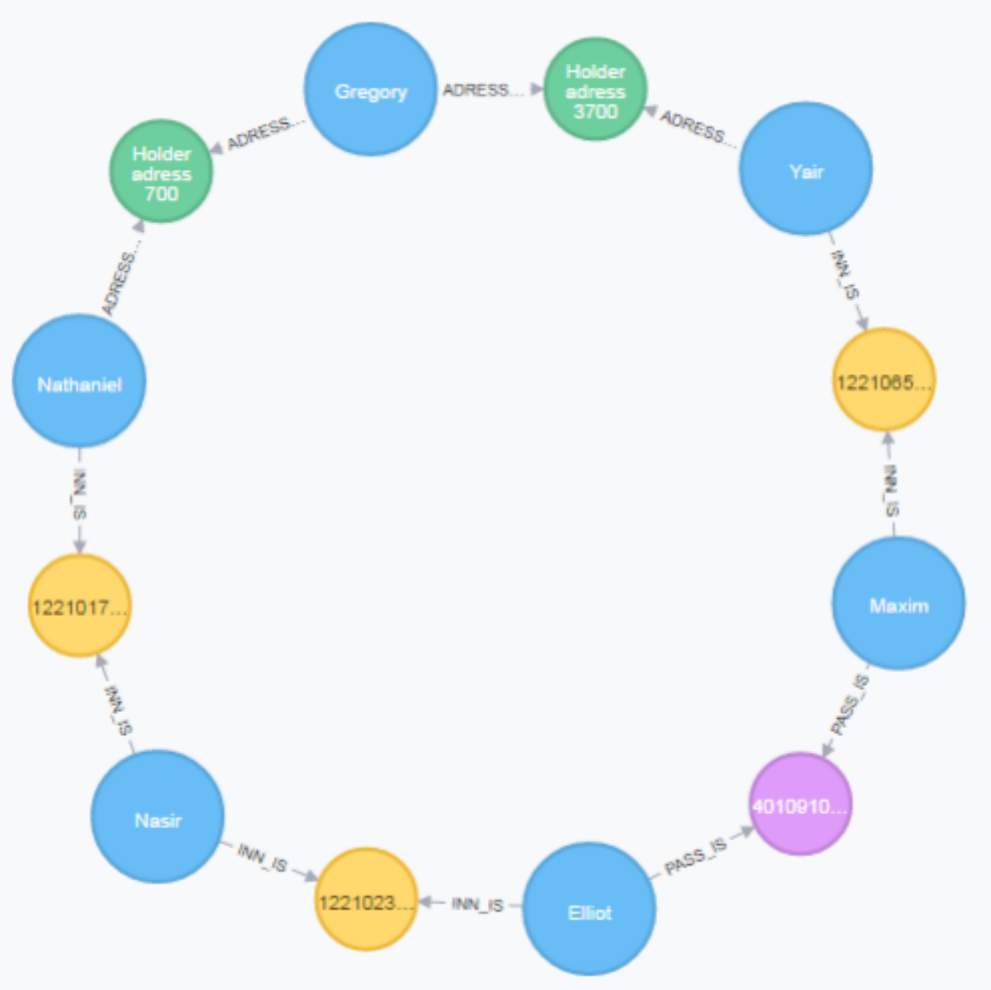

Рисунок 8. Результат в графическом виде.

Примеры не связанного с ИМ банка запроса для демонстрации удобства при работе со связанными данными.

- Запрос наподобие «Выбрать контакты пользователей, которые лайкнули киноактерам, которые снялись в фильмах, в которых звучали

саунтдтреки, которые были написаны музыкантами, которым я поставил лайк» в Neo4j решается тривиально. (Для SQL это гораздо более хлопотное занятие.) Примерно так:

```
MATCH (me:User {userId:123})-[:Like]->(musicants:User)-
[:Author]->(s:Soundtrack)-[:Used]->(f:Film)<-[:Starred]-
(actor: User)<-[:Like]-(u:User) RETURN u
```

- Выбрать вершину с максимальным айди:

```
MATCH (n) RETURN n ORDER BY id(n) DESC LIMIT 1
```

- Выбрать в neo4j все вершины (ограничение по):

```
MATCH n RETURN n LIMIT 25
```

- Получить в neo4j вершину по ID:

```
START n = node(ID[,ID2]) RETURN n
```

- по свойству:

```
MATCH (n) WHERE n.name = "профессия" RETURN n
```

- Выбрать в neo4j такие ноды, у которых есть определенный атрибут:

```
MATCH (n) WHERE HAS (n.name) RETURN n
```

```
MATCH (n) WHERE (n)-[:HYPERONS]->() RETURN n
```

- Удалить в neo4j все вершины и связи. Стоит отметить что нельзя удалить ноду без удаления связей. Возможно при разработке приложения эту операцию выполнять придется весьма часто.

```
MATCH (a) OPTIONAL MATCH (a)-[r1]-() DELETE a, r1
```

- Подсчитать в neo4j количество вершин и связей соответственно:

```
START n=node(*) RETURN count(n)
```

```
START r=relationship(*) RETURN count(r)
```

- Удалить в neo4j изолированные вершины:

```
MATCH n WHERE not (n--()) DELETE n
```

Выводы

Подводя итог, с использованием информационной системы банка в качестве удаленного приложения, можно сделать определенные выводы:

Преимущества графовых информационных систем:

- существует определенные области задач (в особенности напрямую связанные с графами), где эти системы дают значительное преимущество в скорости выполнения запросов;
- разработка запросов для таких информационных систем является более простой задачей, чем в системах, использующих реляционную модель данных;
- существуют определенные области задач, где только графовые системы могут обеспечить полноценное решение;
- отсутствие необходимости в проектировании логической модели базы данных;
- универсальность, ведь в них можно хранить и реляционные, и документарные и сложные семантические данные;
- модель построения БД может меняться и модифицироваться в процессе развития приложения без изменения архитектуры и исходных запросов.

Недостатки графовых информационных систем:

- графовые системы занимают больше физического пространства на хранилищах информации;
- графовые системы требуют больше времени для подготовки (импорта) данных.

Используемая литература

1. Michael Hunger, Ryan Boyd & William Lyon. The Definitive Guide to Graph Databases for the RDBMS Developer, 2016
2. Angles, R. and Gutierrez, C. 2008. Survey of graph database models. ACM Comput. Surv. 40, 1, Article 1, February 2008
3. Графовые базы данных: святой Грааль для разработчиков? Режим доступа: https://habrahabr.ru/post/274383/ (дата обращения 10.01. 2017)
4. Начинаем работать с графовой базой данных Neo4j. Режим доступа: https://habrahabr.ru/post/219441/ (дата обращения 10.01. 2017)
5. Рушкин А.С. Разработка и исследование базы данных для сопровождения удаленного сервиса на основе графовой СУБД Neo4. Режим доступа: http://elib.spbstu.ru/dl/2/v16-1115.pdf/download/v16-1115.pdf?lang=en (дата обращения 15.01. 2017)
6. Документация по языку запросов Cypher. Режим доступа: http://neo4j.com/docs/2.0/cypher-refcard/ (дата обращения 11.01. 2017)
7. Бартенев М.В., Вишняков И.Э. Использование графовых баз данных в целях оптимизации анализа биллинговой информации. Инженерный журнал: наука и инновации, 2013, вып. 11.
8. Бартенев М.В., Вишняков И.Э. Разработка языка запросов к графовому хранилищу биллинговой информации. Инженерный журнал: наука и инновации, 2014, вып. 11.
9. Тихомиров А. Графовая база данных Neo4j. Режим доступа: https://prezi.com/8focrzo7zj9l/neo4j/ (дата обращения 15.01. 2017)
10. Добрянский А. Neo4j VS MySQL. Режим доступа: https://habrahabr.ru/post/258179/ (дата обращения 15.01. 2017)