Минобрнауки России

федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный технологический институт
(технический университет)»

Кафедра систем автоматизированного проектирования и управления

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к практической работе N 8 по курсу

БАЗЫ ДАННЫХ

Разработка графовой базы данных в СУБД Neo4j

ст. преподаватель
Иванов А.А.

2023

Теоретическая часть.

Графовые запросы: введение в Cypher

Cypher - это выразительный (но компактный) язык запросов графовых баз данных. В настоящее время Cypher используется лишь в Neo4j, но представление графов в виде привычных схем делает его идеальным средством программного описания графов. В других графовых базах данных имеются свои средства извлечения данных. Многие из них, в том числе и Neo4j, поддерживают язык запросов SPARQL и императивный язык запросов Gremlin, базирующийся на маршрутах. Но нас интересует представление свойств графов с помощью языка запросов, и по этой причине будем использовать язык Cypher во всех примерах для демонстрации графовых запросов и графовых структур. Cypher является, пожалуй, самым простым графовым языком запросов и послужит основой для изучения графов.

Язык Cypher читабелен и понятен разработчикам, специалистам по базам данных и заказчикам. Простота его использования основывается на сходстве способа описания графов с интуитивным их представлением в виде схем.

Cypher предоставляет пользователю (или приложению, действующему от имени пользователя) возможность задавать шаблон для поиска данных. Проще говоря, можно попросить базу данных «найти что-то похожее на это». Рассмотрим простую модель, изображенную рис. 1.

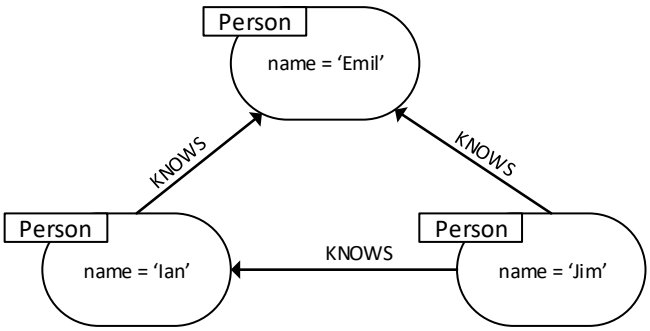

Рисунок - 1 Простая модель графа, изображенная схемой

Приведенный шаблон описывает трех друзей. А вот его эквивалент на Cypher, представленный ASCII-графикой:

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

Этот шаблон описывает маршрут, соединяющий узел с именем jim с двумя другими узлами ian и emil, которые также связаны соединением, идущим от узла ian к узлу emil. ian, jim и emil являются идентификаторами. Идентификаторы позволяют ссылаться на узлы в описании модели и обойти тот факт, что язык запросов имеет только одно измерение (его текст записывается слева направо), в то время как схема графа использует два измерения. Несмотря на то что иногда возникает необходимость в повторении идентификаторов, смысл записи остается ясным.

Запись шаблонов на Cypher выглядит очень естественной, близкой к изображению графов на доске.

Предыдущий Cypher-шаблон описывает структуру простого графа, но не определяет ссылок на конкретные данные в базе данных. Для связывания шаблона с конкретными узлами и взаимосвязями в существующем наборе данных необходимо указать конкретные значения свойств и метки узлов, которые позволят найти соответствующие элементы в наборе данных. Например:

(emil:Person {name:'Emil'})

<-[:KNOWS]-(jim:Person {name:'Jim'})

-[:KNOWS]->(ian:Person {name:'Ian'})

-[:KNOWS]->(emil)

Здесь мы связали узлы с их идентификаторами, используя свойство name и метку Person. Идентификатор emil, например, связан с узлом в наборе данных с меткой Person и со свойством name, имеющим значение Emil. Прикрепление части шаблона к реальным данным в этом случае является обычной практикой для языка Cypher, в чем мы еще не раз убедимся в следующих разделах.

Спецификация на примере

Самым интересным в схемах графов является присутствие конкретных экземпляров узлов и взаимосвязей, а не классов или архетипов. Даже очень большие графы можно изображать с помощью меньших подграфов, состоящих из реальных узлов и взаимосвязей. Другими словами, для описания графов обычно используется спецификация на примере.

Шаблоны графов, являются основой Cypher. Запросы на Cypher прикрепляют одну или несколько частей шаблона к определенным местам графа с помощью предикатов, а затем перемещают незафиксированные части, пытаясь найти соответствие.

Точки фиксации в реальном графе, с которым связаны некоторые части шаблона, определяются в Cypher метками и предикатами свойств, включенными в запрос. В большинстве случаев Cypher использует метаинформацию о существующих индексах, ограничениях и предикатах для автоматического их определения. Но иногда ему требуются дополнительные подсказки.

Как и большинство языков запросов, Cypher состоит из фраз. Простейшие запросы состоят из фразы MATCH и следующей за ней фразы RETURN. Следующий пример запроса на Cypher использует эти фразы для поиска прямых друзей пользователя Jim:

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

RETURN b, c

Рассмотрим подробнее каждую из фраз.

MATCH

Фраза MATCH является основой большинства Cypher-запросов. Это часть спецификации на примере. Используя ASCII-символы для представления узлов и взаимосвязей, мы прорисовываем данные, которые нам нужны. Мы прорисовываем в скобках узлы и вычерчиваем взаимосвязи с помощью пар тире с символами больше или меньше (--> и <--). Символы < и > указывают направление взаимосвязей. Между 'тире' располагается имя взаимосвязи, заключенное в квадратные скобки и предваряемое двоеточием. Метки узлов также предваряются двоеточием. Пары ключ-значение для свойств узла (и взаимосвязи) указываются в фигурных скобках (подобно объектам в JavaScript).

В нашем примере запроса мы ищем узел с меткой Person и значением Jim в свойстве name. Результату этого поиска присваивается идентификатор a. Этот идентификатор позволяет в остальной части запроса ссылаться на узел, представляющий пользователя Jim.

Этот начальный узел является частью простого шаблона: (a)- [:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c), который описывает маршрут, включающий в себя три узла, один из которых мы связали с идентификатором a, два других - с b и c. Эти узлы соединены между собой несколькими взаимосвязями KNOWS, как показано на рис. 1.

Этот шаблон теоретически может встретиться в графе несколько раз; если пользователей достаточно много, с этим шаблоном может быть сопоставлено множество взаимосвязей. Для локализации запроса необходимо закрепить некоторые его части в одном или нескольких местах графа. Указав, что поиск ведется от узла с меткой Person и значением Jim в свойстве name, мы прикрепляем шаблон к конкретному узлу в графе - узлу, представляющему пользователя Jim. Затем Cypher сопоставляет остальную часть шаблона с областью графа, непосредственно окружающей точку привязки. Если они существуют, найденным узлам присваиваются другие идентификаторы. Хотя идентификатор a всегда будет привязан к узлу Jim, идентификаторы b и c в процессе выполнения запроса будут связаны с последовательностью узлов.

Описание точки прикрепления шаблона можно также оформить в виде предиката во фразе WHERE.

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

WHERE a.name = 'Jim'

RETURN b, c
```

Здесь искомое свойство мы переместили из фразы MATCH во фразу WHERE. Результат этого запроса будет тем же.

RETURN

Эта фраза определяет, какие узлы, взаимосвязи и свойства в совпавших данных должны быть возвращены клиенту. В нашем примере запроса необходимо вернуть узлы, связанные с идентификаторами b и c. При обходе полученного результата будут получены все совпавшие узлы, связанные с идентификаторами.

Другие фразы языка Cypher

В запросах на Cypher можно использовать другие фразы:

WHERE

Определяет критерии совпадения результатов для фильтрации шаблона.

CREATE и CREATE UNIQUE

Создает узлы и взаимосвязи.

MERGE

Гарантирует, что заданный шаблон будет существовать в графе либо за счет использования уже найденных в графе узлов и взаимосвязей, соответствующих заданным предикатам, либо за счет создания новых узлов и взаимосвязей.

DELETE

Удаляет узлы, взаимосвязи и свойства.

SET

Устанавливает значения свойств.

FOREACH

Вносит изменения в каждый из элементов списка.

UNION

Объединяет результаты двух или более запросов.

WITH

Объединяет части запроса в цепочку и передает результаты от одной части запроса к другой. Работает подобно именованным каналам в Unix.

START

Явно указывает одну или несколько отправных точек, узлов или взаимосвязей в графе. (Фраза START признана устаревшей, и вместо нее рекомендуется явно указывать отправные точки во фразе MATCH.)

Эти фразы покажутся вам знакомыми, особенно если вы работали с SQL. В то же время следует подчеркнуть их отличие, потому что здесь мы имеем дело с графами, а не реляционными наборами данных.

Шаблоны

Шаблоны и сопоставление образцов находятся в самом сердце Cypher, поэтому для эффективного использования Cypher требуется хорошее понимание шаблонов.

Используя шаблоны, вы описываете форму данных, которые вы ищете. Например, в предложении MATCH вы описываете фигуру с шаблоном, а Cypher выяснит, как получить эти данные для вас.

Шаблон описывает данные, используя форму, которая очень похожа на то, как обычно рисуется форма данных графа свойств на доске: обычно это круги (представляющие узлы) и стрелки между ними для представления отношений.

Шаблоны появляются в нескольких местах в Cypher: в предложениях MATCH, CREATE и MERGE и в выражениях шаблонов.

Шаблоны для узлов

Самая простая «форма», которая может быть описана в шаблоне, - это узел. Узел описывается с использованием пары круглых скобок и обычно присваивается имя. Например:

```
(a)
```

Этот простой шаблон описывает один узел и называет этот узел, используя переменную a.

Шаблоны для связанных узлов

Более мощная конструкция - это шаблон, который описывает несколько узлов и отношения между ними. Модели Cypher описывают отношения, используя стрелку между двумя узлами. Например:

```
(a) --> (b)
```

Этот шаблон описывает очень простую форму данных: два узла и одно отношение от одного к другому. В этом примере оба узла называются как a и b соответственно, а отношение «направлено»: оно идет от a до b.

Этот способ описания узлов и отношений может быть расширен, чтобы покрыть произвольное количество узлов и отношения между ними, например:

```
(a)- -> (b) <-- (c)
```

Такая серия связанных узлов и отношений называется «путем».

Обратите внимание, что именование узлов в этих шаблонах необходимо только в том случае, если нужно снова обратиться к тому же узлу, либо позже в шаблоне, либо в другом месте в запросе Cypher. Если это не обязательно, то имя может быть опущено, как указано ниже:

```
(a) --> () <- -(c)
```

Шаблоны для меток

В дополнение к простому описанию формы узла в шаблоне можно также описать атрибуты. Самый простой атрибут, который может быть описан в шаблоне, - это метка, которую должен иметь узел. Например:

```
(a: Person) --> (b)
```

Можно также описать узел, который имеет несколько меток:

```
(a: Person: Admin) --> (b)
```

Указание свойств

Узлы и отношения являются фундаментальными структурами в графе. Neo4j использует свойства для обоих из них, чтобы использовать гораздо более богатые модели.

Свойства могут быть выражены в шаблонах с использованием map-construct: фигурных скобок, окружающих несколько пар клавиш-выражений, разделенных запятыми. Например, узел с двумя свойствами на нем будет выглядеть так:

```
(a{name: 'Игорь', sport: 'Футбол'})
```

Отношения с ожиданиями на это даются:

```
(a) - [{block: false}] -> (b)
```

Когда свойства появляются в шаблонах, они добавляют дополнительное ограничение к форме данных. В случае предложения CREATE свойства будут установлены во вновь созданных узлах и отношениях. В случае предложения MERGE свойства будут использоваться в качестве дополнительных ограничений для формы, которые должны иметь любые существующие данные (указанные свойства должны точно соответствовать любым существующим данным на графике). Если совпадающие данные не найдены, MERGE ведет себя как CREATE, и свойства будут установлены во вновь созданных узлах и отношениях.

Обратите внимание, что шаблоны, предоставленные CREATE, могут использовать один параметр для указания свойств, например: CREATE (node $paramName). Это невозможно с шаблонами, используемыми в других предложениях, поскольку Cypher должен знать имена свойств во время компиляции запроса, так что сопоставление может быть выполнено эффективно.

Шаблоны для отношений

Самый простой способ описать связь - это использовать стрелку между двумя узлами, как в предыдущих примерах. Используя эту технику, вы можете описать, что должна существовать связь и ее направленность. Если вы не заботитесь о направлении отношения, стрелочную голову можно опустить, как показано на примере:

```
(a) -- (b)
```

Как и для узлов, отношениям также могут быть присвоены имена. В этом случае пара квадратных скобок используется для разрыва стрелки, и переменная помещается между ними. Например:

```
(a) - [r] -> (b)
```

Подобно ярлыкам на узлах, отношения могут иметь типы. Чтобы описать отношения с определенным типом, вы можете указать это следующим образом:

```
(a) - [r: REL_TYPE] -> (b)
```

В отличие от меток, отношения могут иметь только один тип. Но если мы хотим описать некоторые данные таким образом, чтобы отношение могло иметь один из множества типов, то все они могут быть перечислены в шаблоне, то все они могут быть перечислены в шаблоне, разделяя их символом | как это:

```
(a) - [r: TIPE1 | TIPE2] -> (b)
```

Обратите внимание, что эта форма шаблона может использоваться только для описания существующих данных (т. Е. При использовании шаблона с MATCH или как выражение). Он не будет работать с CREATE или MERGE, так как невозможно создать отношения с несколькими типами.

Как и в случае с узлами, имя отношения всегда можно опустить, как показано на примере:

```
(a) - [: REL_TYPE] -> (b)
```

Сравнение шаблонов с переменной длиной

Сравнение шаблонов с переменной длиной в версиях 2.1.x и более ранних версий не обеспечивает единства отношений для шаблонов, описанных в одном предложении MATCH. Это означает, что запрос, такой как следующий: MATCH (a) - [r] -> (b), p = (a) - [*] -> (c) RETURN *, relationship (p) AS rs могут включать r как часть набора rs. Такое поведение изменилось в версиях 2.2.0 и более поздних версий таким образом, что r будет исключен из набора результатов, поскольку это лучше соответствует правилам уникальности отношений. Если у вас есть шаблон запроса, который должен восстанавливать отношения, а не игнорировать их, как обычно диктуют правила уникальности отношения, вы можете выполнить это, используя несколько предложений соответствия, следующим образом: MATCH (a) - [r] -> (b)

MATCH p = (a) - [*] -> (c) RETURN *, отношения (p). Это будет работать во всех версиях Neo4j, которые поддерживают предложение MATCH, а именно 2.0.0 и новее.

Вместо описания длинного пути, использующего последовательность многих описаний узлов и отношений в шаблоне, многие отношения (и промежуточные узлы) можно описать, указав длину в описании отношения шаблона. Например:

```
(a) - [*2] -> (b)
```

Это описывает граф трех узлов и двух отношений, все в одном пути (путь длины 2). Это эквивалентно:

```
(a)- -> ()- -> (b)
```

Также может быть указан диапазон длин: такие шаблоны отношений называются «отношениями с переменной длиной». Например:

```
(a) - [*3..5] -> (b)
```

Это минимальная длина 3 и максимум 5. Она описывает граф из 4 узлов и 3 отношений, 5 узлов и 4 отношений или 6 узлов и 5 отношений, все связанные друг с другом в одном пути.

Любая граница может быть опущена. Например, чтобы описать пути длиной 3 или более, используйте:

```
(a) - [*3..] -> (b)
```

Чтобы описать пути длиной 5 или менее, используйте:

```
(a) - [* .. 5] -> (b)
```

Обе границы можно опустить, позволяя описывать пути любой длины:

```
(a) - [*] -> (b)
```

В качестве простого примера возьмем более сложный граф, представленный на рис. 2 и запрос ниже:

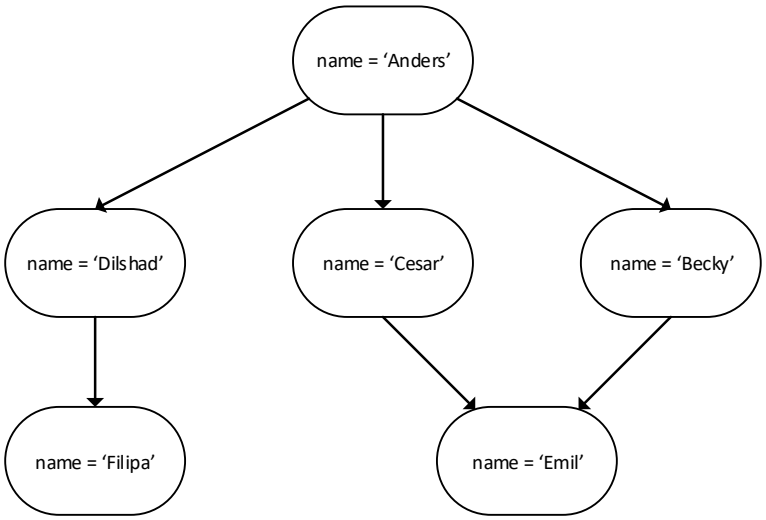

Рисунок – 2. Более сложный пример графа

Запрос.

```
MATCH (me) - [: KNOWS*1..2] - (remote_friend)

WHERE me.name = 'Filipa'

RETURN remote_friend.name
```

В результате получим:

```
remote_friend.name

2 строки

"Dilshad"

"Anders"
```

Этот запрос находит данные на графе, который соответствует форме, в частности, узлу (с именем name «Filipa»), а затем связанным с KNOWS узлам, одному или двум ударам. Это типичный пример поиска друзей первой и второй степени.

Обратите внимание, что отношения переменной длины не могут использоваться с CREATE и MERGE.

Присвоение переменных пути

Как описано выше, ряд связанных узлов и отношений называется «путем». Cypher позволяет указывать пути с использованием идентификатора, как показано на примере:

```
p = (a) - [*3..5] -> (b)
```

Вы можете сделать это в MATCH, CREATE и MERGE, но не при использовании шаблонов в качестве выражений.

Изучив основу языка Cypher, можно приступить к выполнению лабораторной работы.

Цель работы: изучение особенностей работы с графовыми базами данных на примере СУБД Neo4j.

Разработка базы данных социальной сети.

Используя графовую СУБД Neo4j, разработайте базу данных социальной сети. Узлами социальной сети являются персоны, имеющие следующие атрибуты: ФИО, пол, возраст, город. Второй тип узлов - группы по интересам (например: спорт, игры, компьютеры). Семантика связи между узлами – дружеские отношения соответствующих персон («А является другом Б») и членство персоны в группе («персона А состоит в группе G»).

Разработка запросов к графам.

Разработайте и протестируйте запросы на выборку данных из созданной графовой базы данных:

1) Выдать упорядоченный список ФИО персон.

2) Выдать список ФИО мужчин с указанием возраста, упорядоченный по убыванию возраста.

3) Выдать упорядоченный список ФИО друзей персоны заданными ФИО.

4) Выдать упорядоченный список ФИО друзей друзей персоны заданными ФИО.

5) Выдать упорядоченный по алфавиту список ФИО персон, в котором для каждой персоны указано количество друзей.

6) Выдать упорядоченный список групп социальной сети.

7) Выдать упорядоченный список групп персоны с заданными ФИО.

8) Выдать список групп социальной сети с указанием количества членов каждой группы, упорядоченный по убыванию количества членов группы.

9) Выдать список ФИО персон, в котором для каждой персоны указано количество групп, в которые она входит, упорядоченный по убыванию количества групп.

10) Выдать общее количество групп, в которых состоят друзья персоны с заданными ФИО.

В отчет по лабораторной работе необходимо включить скрипт создания БД, сам граф, а также тексты запросов с результатами их выполнения (не более 10 записей по каждому из запросов).