

ЦИФРОВАЯ
КУЛЬТУРА
УНИВЕРСИТЕТ ИТМО

Neo4j

Высшая Школа Цифровой Культуры
Университет ИТМО

dc@itmo.ru

Содержание

1	Введение в Neo4j	2
2	Проектирование графовых баз	4
3	Работа с данными	6

1 Введение в Neo4j

Графовые базы данных используют графовую модель для представления и хранения данных. Если в реляционной модели данные хранятся в виде жесткой структуры с предопределенной схемой, то в графовой базе данных никакой предопределенной схемы нет. Скорее сама схема является отражением тех данных, которые поступили в базу. Чем разнообразнее данные – тем сложнее схема.

Самым популярным представителем этого семейства баз является: Neo4j. Загрузить на свой компьютер версию Neo4j можно с официального сайта, который указан на слайде. Кроме того, для создания небольших баз можно

Графовая база Neo4j

Neo4j - наиболее яркий представитель
графовых баз данных.

<https://neo4j.com/>

- официальный сайт Neo4j

<https://console.neo4j.org/>

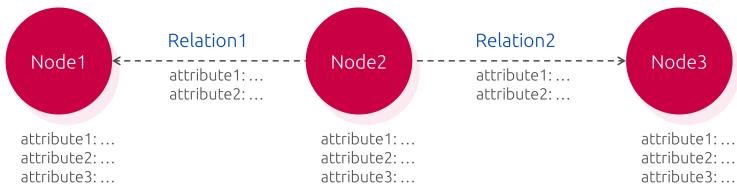
- консольная веб-версия



попробовать консольную веб-версию, адрес которой также указан на слайде.

Графовая модель позволяет хранить сущности и отношения между ними. Сущности моделируются узлами графа, которые имеют свои атрибуты. Отношения моделируются ребрами. Ребра имеют направления, а также могут иметь свои атрибуты. Структура графа позволяет один раз записать данные, а затем интерпретировать их разными способами в соответствии с отношениями. Кроме того, к графу легко добавляются новые узлы и новые отношения.

Какие данные могут быть представлены в виде графа?



Рассмотрим пример данных, имеющих графовую структуру. Это схема метро Санкт-Петербурга. Еще недавно эта схема имела следующий вид, который показан на слайде. Как известно, в 2018 году были открыты новые

станции метро: Новокрестовская и Беговая. Заметьте, что добавление новых

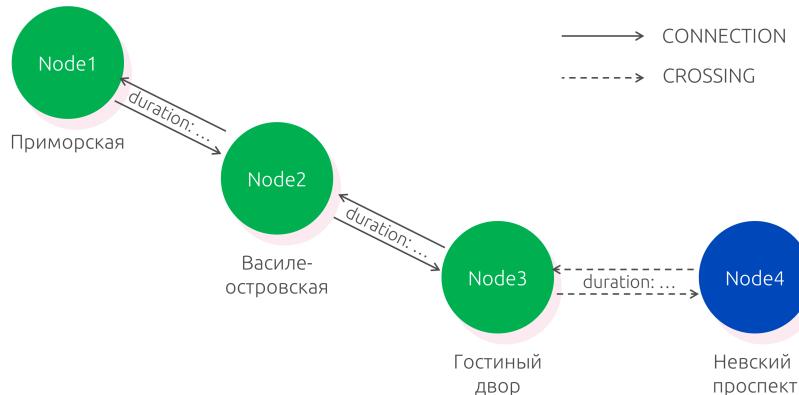
Схема метро Санкт-Петербурга 2018



станций никак не затронуло существующие части схемы. Всего лишь произошло добавление новых узлов и ребер к графу.

Рассмотрим фрагмент небольшой графовой базы данных, содержащий только несколько станций метро. В базе данных, которую вы видите на экране, присутствуют узлы графа, представляющие станции метро. У каждого узла есть свойства. В нашем случае – это одно свойство `Name`. Узлы графа соединены ребрами. Ребра двух типов: `CONNECTION` и `CROSSING`. Содержательно, первый тип связи означает, что станции, которые она соединяет, связаны одним перегоном (например, между метро Василеостровская и Приморская один перегон). Связь типа `CROSSING` означает, что возможна пересадка с одной станции на другую (например, со станции Гостиный двор возможна пересадка на станцию Невский проспект). У каждого типа ребер есть и дополнительная нагрузка – свойство `Duration`, которое в первом случае означает

Из чего состоит графовая база данных?



нительная нагрузка – свойство `Duration`, которое в первом случае означает

время, за которое поезд метро преодолевает соответствующий перегон, а во втором случае – время, которое требуется пассажиру, чтобы перейти с одной станции на другую. Конечно, и в традиционных реляционных СУБД тоже существуют возможности для хранения связей между однотипными и разнородными объектами. Как правило, такого рода связи подкрепляются ссылочными (*reference*) и само-ссылочными (*self-reference*) правилами целостности. Однако добавление новых связей и изменение существующих в такие схемы является очень тяжелой операцией, требующей перемещения большого количества данных. В графовых базах добавление нового отношения является примитивной операцией, не затрагивающей все остальные данные.

В графовых базах запрос на выборку данных из графа принято называть обходом графа. Обход графа происходит быстрее, чем в реляционных базах потому, что связи (отношения) между узлами не надо вычислять, так как они напрямую присутствуют в структуре хранения графа.

2 Проектирование графовых баз

Как создать графовую базу данных? Существует много разнообразных графовых баз данных со своими встроенными командами для создания базы. Но, тем не менее, как бы они не отличались друг от друга, их объединяет то, что их основными командами при создании базы являются команды, позволяющие задавать сущности с атрибутами и отношения между ними.

Для создания графовых баз и манипуляций над данными используются специализированные языки. В СУБД Neo4j для этих целей используется язык *Cypher*. Посмотрим, как используя *Cypher* можно задать графовую базу на примере создания фрагмента базы метрополитена.

Для создания отдельного узла используется специальная конструкция, которая позволяет задать узел в графе, его тип (*station*) и задать его свойство *name*. На экране можно видеть пример создания двух узлов, представляющих станции метро.

Пример создания узлов графа

(*Vasileostrovskaya:station{name: 'Vasileostrovskaya'}*)

(*NevskyProspekt:station{name:'Nevsky Prospekt'}*)



Следующая конструкция на экране используется для задания отношений. Она позволяет указать имена соединяемых узлов, установить направленную связь между ними и указать имя отношения (**CONNECTION**). В приве-

Пример создания отношений графа

(*Vasileostrovskaya*)-[:**CONNECTION**]->(*NevskyProspekt*)

(*NevskyProspekt*)-[:**CONNECTION**]->(*Vasileostrovskaya*)

денном примере необходимо было создать отношения между узлами в двух направлениях, потому что поезда в метрополитене между этими станциями двигаются в двух направлениях. Узлам должны быть известны входящие и исходящие отношения, которые можно обойти в двух направлениях. Отношения являются полноценными элементами графовых баз и, собственно, ценность этих баз данных в основном обусловлена наличием отношений.

Отношения имеют не только тип, начальный и конечный узел, но и свои собственные свойства. Используя эти свойства, в отношение можно внести

Задание дополнительных свойств отношений

(*Vasileostrovskaya*)-[:**CONNECTION {duration:4}**]->

(*NevskyProspekt*)

дополнительную информацию, например, время прохождения перегона или перехода с одной станции метро на другую. Пример на экране демонстрирует, как можно задавать отношениям дополнительные свойства.

И, наконец, следующий пример демонстрирует команду **CREATE**, позволяющую задать три станции метрополитена и отношения между ними.

Пример создания графа

```
CREATE (Vasileostrovskaya:station{name:'Vasileostrovskaya'}),  
(NevskyProspekt:station{name:'Nevsky Prospekt'}),  
(GostinyDvor:station{name:'Gostiny Dvor'}),  
(Vasileostrovskaya)-[:CONNECTION {duration:4}]->(NevskyProspekt),  
(NevskyProspekt)-[:CONNECTION {duration:4}]->(Vasileostrovskaya),  
(NevskyProspekt)-[:CROSSING {duration:2}]->(GostinyDvor),  
(GostinyDvor)-[:CROSSING {duration:2}]->(NevskyProspekt)
```



Свойства узлов и отношений можно индексировать. Назначение индексов в графовых базах – быстрый поиск стартовой точки для обхода графа. В приведенном ранее примере имеет смысл построить индекс по полю `name` для узлов графа.

Создать индекс можно командой `CREATE INDEX`. На экране вы можете

Индексация свойств графа

CREATE INDEX ON :station (name)

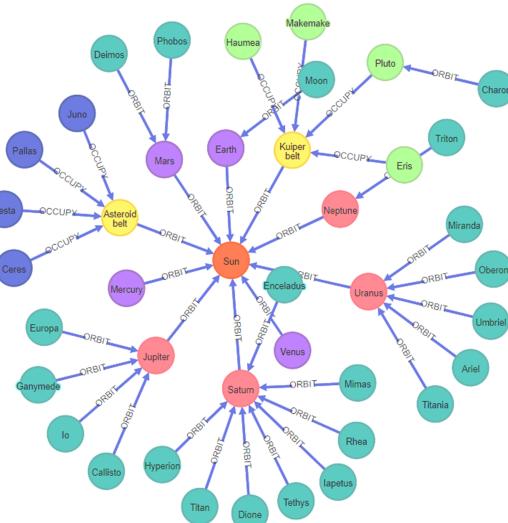


видеть пример построения такого индекса по полю `name` для узлов, представляющих станции метро.

3 Работа с данными

Специализированные языки, ориентированные на работу с графиками, позволяют не только создавать графы, но и запрашивать свойства узлов, обходить графы и перемещаться по узлам и отношениям. Продемонстрируем, как можно строить запросы на языке `Cypher` на примере достаточно простой графовой базы, представляющей планеты солнечной системы.

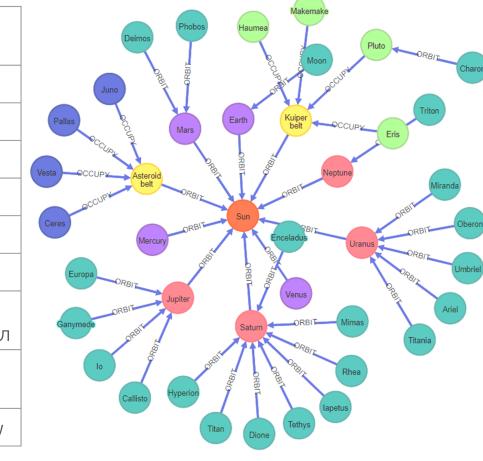
База данных «Планеты солнечной системы»



В базе есть 5 типов узлов и 2 типа отношений. У каждого объекта базы (узла или отношения) есть атрибут `name`, а у некоторых узлов есть свойство `dist`.

Объекты базы

Категория объекта	Тип объекта	Описание объекта
Узел	Star	звезда
	TerrestrialPlanet	планеты земной группы
	GiantPlanet	планеты-гиганты
	DwarfPlanet	карликовые планеты
	Planet	другие планеты
	Asteroid	астероиды
	Region	области Солнечной системы, объединяющие группы малых тел
Отношение	ORBIT	один из объектов вращается вокруг другого
	OCCUPY	указывает на вхождение в группу



Рассмотрим, как в языке `Cypher` выглядит простейший запрос на выборку данных. Для выполнения запросов в языке `Cypher` используется команда `MATCH`, которая во многом похожа на оператор `SELECT` в языке `SQL`. Конструк-

Синтаксис простейших запросов

`MATCH <data_filter>`

`RETURN <projection>`

`[ORDER BY ...]`



ция, которая записана непосредственно после ключевого слова `MATCH` является условием/фильтром для выборки, а конструкция после слова `RETURN` – проекцией, т.е. указывает какие именно поля и значения должны быть выведены в результате выборки. Конструкция `ORDER BY`, как обычно, используется для указания порядка сортировки.

Вернемся к нашей базе и попробуем написать запрос, выбирающий данные планеты-гиганта Сатурн. Есть два возможных способа написания этого запроса: без конструкции `WHERE` и с конструкцией. На слайде продемонстрированы оба варианта. Как в первом, так и во втором случае мы указываем, что нас интересуют узлы типа `GiantPlanet` с значение атрибута `name`, указывающим на Сатурн. В качестве результата запроса будут выведены все

Пример запроса (выборка планеты-гиганта `Saturn`)

```
MATCH (p:GiantPlanet {name: 'Saturn'}) RETURN p
```

или

```
MATCH (p:GiantPlanet)
```

```
WHERE p.name = 'Saturn'
```

```
RETURN p
```



сведения об узлах удовлетворяющих этим условиям. Но такой узел в базе найдется только один. Если выполнить этот запрос в упомянутом ранее консольном веб-интерфейсе вы должны увидеть то, что сейчас изображено на экране.

```
Query:  
MATCH (p:GiantPlanet {name: 'Saturn' }) RETURN p
```

p
(18:GiantPlanet {dist:1434, name:"Saturn"})

```
Query took 17 ms and returned 1 rows. Result Details
```

```
MATCH (p:GiantPlanet {name: 'Saturn' }) RETURN p
```

Усложним запрос и выведем имена планет, которые вращаются вокруг Сатурна. Для этого напишем следующий запрос, который также может быть записан в двух вариантах. Как в первом, так и во втором варианте напишем, что нас интересуют узлы типа `GiantPlanet` с значение атрибута `name`, указывающим на Сатурн и те узлы, которые с ним связаны отношением типа `ORBIT`. В качестве результата запроса выведем значения атрибута `name` этих узлов.

Пример запроса (выборка планет, вращающихся вокруг Сатурна)

```
MATCH (p:GiantPlanet {name: 'Saturn'}) <-[ORBIT]-(orbit_node)
RETURN orbit_node.name
```

или

```
MATCH (p:GiantPlanet) <-[ORBIT]-(orbit_node)
WHERE p.name = 'Saturn'
RETURN orbit_node.name
```



Результат такого запроса отображен на экране.

Query:

```
MATCH (p:GiantPlanet {name: 'Saturn' }) <-[ORBIT]-(orbit_node) RETURN orbit_node.name
```

orbit_node.name
Iapetus
Hyperion
Titan
Rhea
Dione
Tethys
Enceladus
Mimas

Query took 20 ms and returned 8 rows. [Result Details](#)

```
MATCH (p:GiantPlanet {name: 'Saturn' }) <-[ORBIT]-(orbit_node) RETURN orbit_node.name
```

Усложним задание и выведем названия планет, которые врачаются вокруг всех планет-гигантов и названия самих планет-гигантов. Сам запрос

Результат запроса

```
MATCH (p:GiantPlanet) <-[:ORBIT]-  
(orbit_node)  
RETURN p.name, orbit_node.name  
ORDER BY p.name, orbit_node.name
```

p.name	orbit_node.name
Jupiter	Callisto
Jupiter	Europa
Jupiter	Ganymede
Jupiter	Io
Neptune	Triton
Saturn	Dione
Saturn	Enceladus
Saturn	Hyperion
Saturn	Iapetus
Saturn	Mimas
Saturn	Rhea
Saturn	Tethys
Saturn	Titan
Uranus	Ariel
Uranus	Miranda
Uranus	Oberon
Uranus	Titania
Uranus	Umbriel

стал проще. Мы не указываем название планеты, а задаем только тип исходных узлов – `GiantPlanet` (планеты-гиганты). В качестве результата запроса выдаются названия планет-гигантов и планет, которые с ними связаны отношением `ORBIT`.

Кроме того, можно, как в обычных SQL-запросах упорядочить результат при выводе с помощью конструкции `ORDER BY`.

Поиск узлов и их прямых отношений не представляет особого интереса, так как это же можно делать в реляционных базах данных. Настоящая мощь графовых баз данных проявляется в тех случаях, когда необходимо обойти граф на произвольную глубину начиная с заданной стартовой точки. Следующий пример демонстрирует, как можно найти все узлы, до которых можно добраться из узла с названием “Солнце” используя связи типа `ORBIT` (причем названия узлов будут выводиться в лексикографическом порядке).

Запрос похож на предыдущий. Отличается только стартовая точка и глубина обхода. Для того, чтобы написать запрос, который выполнит обход на произвольную глубину используется символ `*`, который указывает, что отношение `ORBIT` может встретиться во время обхода графа произвольное количество раз.

Пример обхода графа на произвольную глубину

```
MATCH (p:Star)<-[:ORBIT*]-(orbit_node)
WHERE p.name = 'Sun'
RETURN orbit_node.name
ORDER BY orbit_node.name
```



В графовых базах немало интереснейших возможностей. В них активно используются разнообразные алгоритмы обхода графов, позволяющие находить пути между заданными узлами. В частности, можно определить, есть ли несколько путей, найти все пути или кратчайший путь. Разумеется, в рамках данного курса мы не можем осветить все подробности работы с Neo4j и языком запросов *Cypher* и, поэтому, как обычно, отсылаем любознательных слушателей к документации <https://neo4j.com/docs/>.