МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение  
 высшего образования   
**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Кафедра вычислительных технологий**

**КУРСОВАЯ РАБОТА**

**РАЗРАБОТКА …**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.А. Пасько

(подпись, дата)

Факультет компьютерных технологий и прикладной математики курс 3

Направление 02.03.02 – «Фундаментальная информатика и  
информационные технологии»

Научный руководитель доц.,   
канд.тех. наук, доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Т.А. Приходько

(подпись, дата)

Нормоконтролер преп.,  
канд.тех. наук \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.Е. Полупанова

(подпись, дата)

Краснодар 2017

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc495569176)

[1. ГРАФОВЫЕ NOSQL БАЗЫ ДАННЫХ 5](#_Toc495569177)

[1.1 Характеристики графовых СУБД 5](#_Toc495569178)

[1.2. Архитектура и способы работы с Neo4j 6](#_Toc495569179)

[1.3 Язык запросов Cypher 7](#_Toc495569180)

[2. РАЗРАБОТКА ПРИЛОЖЕНИЯ НА ОСНОВЕ NOSQL Neo4j 9](#_Toc495569181)

[3. ВЫПОЛНЕНИЕ АНАЛИЗА ДАННЫХ 10](#_Toc495569182)

[ЗАКЛЮЧЕНИЕ 11](#_Toc495569183)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 12](#_Toc495569184)

ВВЕДЕНИЕ

В мире технологий баз данных существует два основных направления: реляционные (SQL) и нереляционные (NoSQL) базы данных. Различия между ними значительные. Рассмотрим каждую из них.

Первое направление – реляционная база данных (SQL) была введена Эдгаром Коддом 1974-м году. Ее основная идея – это разделение данных на концептуальное и логическое проектирование. Благодаря простоте использования, она получила широкую популярность среди бизнес-приложений, но несмотря на то, что реляционные хранилища обеспечивают наилучшее сочетание простоты, устойчивости, совместимости, ее показатели не всегда выше, чем у аналогичных систем. В связи с этим, в последнее время стала очень распространенной базой данных NoSQL.

Термин NoSQL был применен Карло Строцц в 1998 году к его небольшой реляционной СУБД, которая не использовала SQL для манипулирования данными.

С 2009 года термин «NoSQL» стал применяться уже для обозначения растущего числа распределенных систем управления данными, которые отказывались от поддержки ACID-транзакций (atomicity, consistency, isolation, durability – атомарность, согласованность, изолированность, постоянство хранения).

Как правило, существует несколько причин, по которым рассматривают возможность использования базы данных NoSQL:

* Гибкость модели, в том числе возможность вносить масштабные
* изменения в модель за короткое время.
* Высокая производительность для наборов данных с большим
* количеством записей и связей между ними.
* Низкая сложность в создании, заполнении и обслуживании.

Термин NoSQL включает в себя множество разных систем такие как: Cassandra, MongoDB, CouchDB, ElasticSearch, Neo4j и др. В данной работе будет рассматриваться графовая СУБД Neo4j; в среде будет реализована база данных, на примере которой будут рассмотрены возможности декларативного языка Сypher, а также теоретический анализ полученных данных.

Целью работы является изучение принципов функционирования графовых БД на примерах, проведение элементарного анализа данных, визуализация полученных результатов.

1. ГРАФОВЫЕ NOSQL БАЗЫ ДАННЫХ

В последнее время, данные, с которыми мы имеем дело, становятся все более взаимосвязанными, и для их удержания и обработки требуются специальные хранилища, одним из наиболее распространенных нереляционных хранилищ является графовая БД.

Интерес к графовым базам данных возник с конца 1980-х годах, однако всем известно, что теория графов была впервые введена Эйлером в 18-м веке, и активно изучалась и наблюдалась исследователями многих областей. Тем не менее, только в последние несколько лет теория графов приобрела особо значимый вес в области хранения информации.

Исходя из названия, основная модель данных представляется классическим математическим графом. Графовые базы данных позволяют хранить сущности и отношения между ними. Сущности моделируются узлами, которые имеют свойства. Узел интерпретируется как экземпляр объекта в приложении. Отношения моделируются ребрами, которые могут иметь свойства. Ребра имеют направление; узлы организованы в соответствии с отношениями. Все это позволяет находить требуемые шаблоны среди узлов. Организация графа позволяет один раз записать данные, а затем интерпретировать разными способами в соответствии с отношениями.

Система управления графовыми базами данных поддерживает методы создания (Create), чтения (Read), изменения (Update) и удаления (Delete) или CRUD.

Далее на рисунке 1.1 приведен графический обзор некоторых графовых баз

данных из представленных сегодня на рынке, основанных на разных моделях хранения и обработки информации.

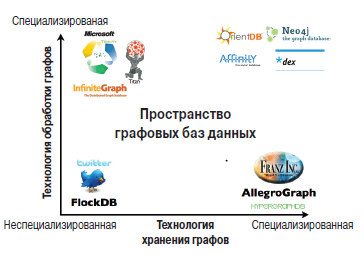


Рисунок 1.1 - графический обзор графовых баз данных

Графы чрезвычайно полезны при анализе самых разных наборов данных в таких областях, как наука, государственное управление, бизнес и т.д. Реальный мир, в отличие от основанной на шаблонах устаревшей модели реляционных баз данных, разнообразен и взаимосвязан: в одних местах равномерно упорядочен, в других становится случайным и нерегулярным. Графовые базы данных помогли решить важнейшие проблемы в области социальных сетей, управления корпоративными данными, геопространством и некоторых других (рис. 1.2). Что качается их практического внедрения, то, работая с графовыми СУБД, коммерческий успех получили такие передовые компании, как Facebook, Google , Twitter и другие.



Рисунок.1.2. Области применения графовых баз данных

1.1 Характеристики графовых СУБД

Одним из главных достоинств графовых моделей БД является то, что с ними можно обращаться как с простым графом, т.е осуществлять естественную реализацию графовых операций (поиска путей, выделения сообществ и других). Также они имеют гибкую схему данных, позволяющую унифицировать хранение разнородных объектов. Одной из особенностей графовой базы данных является то, что она позволяет использовать наивную структуру своих данных, т.е смежность без индексов и при этом каждый узел содержит прямые ссылки на смежные с ним узлы. Это очень эффективный способ обработки информации, ведь каждый узел представляет собой микроиндекс, ссылающийся на соседние узлы, а это значительно снижает затраты, если сравнивать с использованием глобальных индексов.Понятно, что время выполнения запроса будет зависеть неот общего размера графа, а от размера части графа, участвовавшейв поиске. Нативные системы управления графовыми базами данных используют только глобальные индексы для связывания узлов. Обработка индексов обычных графов занимает достаточно большие вычислительные ресурсы. Поэтому можно отметить, что нативное хранение баз данных - это ключевой фактор в части скорости и эффективности.

Вообще, независимо какая конкретно графовая модель будет выбрана, иллюстрируемые идеи и примеры, рассматриваемые в курсовой, применимы ко многим графовым базам данных.

1.2. Архитектура и принципы работы Neo4j

При разработке проектов, которые основаны на графовых базах данных, выделяют несколько вариантов построения архитектуры приложения. Эти варианты не имеют особых отличий друг от друга, но все же различны. Выбор зависит от конкретной базы данных. В работе будут описаны варианты, которые доступны Neo4j.

**Neo4j** – это графовая система управления базами данных с открытым исходным кодом, реализованная на языке Java. Была образована американской компанией Neo Technology 2003 году. На 2017 год **Neo4j** считается самой распространённой графовой СУБД (по рейтингу сайта DB-Engines).

На сегодняшний день, большинство баз данных представляются в качестве серверов, обращение к которым происходит через клиентскую библиотеку.

Отличительной чертой базы данных Neo4j является то, что она может работать как во встроенном, так и в серверном режимах.

Во встроенном режиме Neo4j выполняется в одном процессе с приложением. Данный режим подходит для приложений, устройств и также для внедрения в приложение собственных серверов.

Рассмотрим некоторые преимущества встроенного режима:

1. малое время отклика - приложение обращается к базе данных напрямую, а не через сеть;
2. широкий выбор программных интерфейсов - имеется широкий доступ к интерфейсам, которые позволяют создавать и выполнять запросы (интерфейс Core API, фрейм-ворк Traversal, язык запросов Cypher)
3. жизненный цикл базы данных – приложение отвечает за управление;
4. всем жизненным циклом базы данных, включая ее запуск и безопасное закрытие;

Базы данных, которые работают во встроенном режиме также допустимо объединять в кластеры для обеспечения высокой доступности и масштабировать точно так же, как базы данных в серверной версии, но также можно запускать смешанный кластер встроенных и серверных экземпляров, т.е кластеризация выполняется на уровне базы данных, а не на уровне сервера.

Серверный режим является самым распространенным способом запуска базы данных в Neo4j. Рассмотрим некоторые преимущества этого варианта запуска:

1. доступность интерфейса REST API – сервер предоставляет расширенный программный интерфейс REST API, позволяющий клиентам отправлять запросы в формате JSON по протоколу HTTP. Ответы включают в себя документы в JSON формате, содержащие гиперссылки, расширяющие функции набора данных. Интерфейс REST API поддерживает расширение конечными пользователями и выполнение запросов на Cypher;
2. независимость от платформы – поскольку поддерживается доступ посредством документов в формате JSON, передаваемых по протоколу HTTP, сервер Neo4j доступен клиенту, работающему практически на любой платформе. Все, что требуется, это клиентская библиотека HTTP;
3. независимость от масштабирования – при работе базы данных Neo4j в режиме сервера, кластер базы данных можно масштабировать независимо от кластера сервера приложений;
4. изоляция от сборки мусора – в серверном режиме база данных Neo4j защищена от любых неблагоприятных влияний, связанных со сборкой мусора. Конечно, работа базы данных Neo4j приводит к образованию мусора, но ее влияние на сборку мусора тщательно контролируется, и она настроена так, чтобы смягчить любые серьезные побочные эффекты. Но база данных Neo4j обеспечивает использование серверных расширений, позволяя запускать на сервере произвольный Java-код, а выполнение серверных расширений может повлиять на сборку мусора сервера;

Архитектура Neo4j дает возможность управлять информацией, хранить и производить обход узлов и связей. В Neo4j отношения являются важной составляющей, как и в любой другой графовой СУБД, которые также позволяет создавать связи между сущностями (рис.1.3).



Рисунок 1.3. -

Neo4j оперирует следующими объектами:

* вершины (nodes) – и их помощью представляются сущности графа, также, в зависимости от отношений в графе, могут быть использованы для представления связи. Самый простой граф представляет собой одну вершину. Вершина может не иметь, либо иметь одно или более именованных значений, которые указываются в виде свойств;
* связи (relationships) – соединяют узлы-начальный и конечный. Связи, также, как и узлы, могут иметь свойства.
* свойства (prosperties) – именованные значения, где имя – это строка. Поддерживаемые значения: числовые, строковые, двоичные, списки предыдущих типов;
* метки (labels) – представляются в виде графов, которые были сгруппированы в наборы. Все узлы, помеченные одной меткой, принадлежит к одному набору. Они намного упрощают написание запросов к базе. Вершина может быть помечена любым количеством меток. Метки используются для задания ограничений и добавления индексов для свойств.

Графовая СУБД хранит данные в графе, которая, являясь довольно гибкой структурой данных, она способна изящно представить любую информацию в высоко доступной форме.

В графическом интерфейсе можно видеть, что вся информация отображается в блоке Database Information, где данные, связанные с узлами и метками, хранятся в части Node Labels, а со связями - в Relationship Types, это дает возможность в любой момент восстановить нужную нам информацию о связях или узлах.

В СУБД Neo4j реализован графический интерфейс, в котором могут быть удобно представлены графовые данные. Но данные могут быть представлены не только в виде графа, но и в строковом представлении.

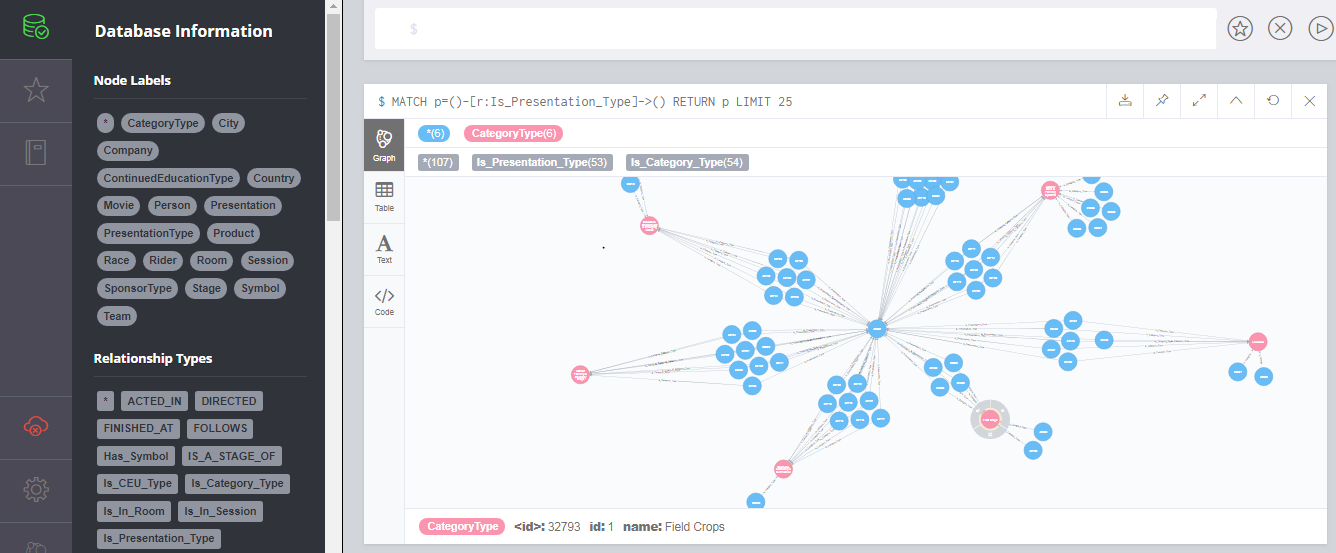


Рисунок 1.4 «Интерфейс Neo4j»

1.3 Язык запросов Cypher

Cypher - декларативный язык. Он фокусируется на ясности выражения того, что необходимо извлечь из графа, а не в том, как получить это. Это и 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 (появление новых индексов и т.д.).

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;

Графовая база данных хранит информацию в специальном собственном формате, такой подход, в сравнении с моделированием графовой базы данных средствами реляционной СУБД, позволяет применять дополнительную оптимизацию в случае данных с более сложной структурой. Также утверждается о наличии специальных оптимизаций для SSD-накопителей, при этом для обработки графа не требуется его помещение целиком в оперативную память вычислительного узла, таким образом, возможна обработка достаточно больших графов.

Компоненты графовой базы данных — узлы и ребра. Они могут быть дополнены собственным набором полей. Модель такой БД схематично изображена на рисунке 1.5.

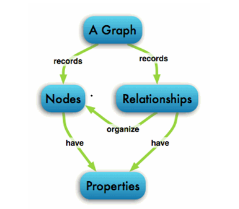


Рисунок 1.5 -

Чтобы начать работать с Neo4j, необходимо создать или связать данные (информацию) с базой данных. Но в Neo4j можно это сделать разными способами, например, запросами Cypher непосредственно в интерфейсе базы данных или импортирования .csv файлов. Для описанной в данной работе системы выбран первый вариант, хотя второй значительно выигрывает по скорости, как показала практика.

Также существует 4 способа для более широкого взаимодействия с Neo4j:

* Использование API в JVM-дружественном языке, таком как Java, Scala или Clojure.
* Использование интерфейса веб-службы REST.
* Использование [языка Cypher](http://docs.neo4j.org/chunked/stable/cypher-query-lang.html) в веб-интерфейсе пользователя.
* Использование языка Cypher в командной оболочке Neo4J.

2. РАЗРАБОТКА ПРИЛОЖЕНИЯ НА ОСНОВЕ NOSQL Neo4j

В качестве примере в работе решена задача исследования конференции. Научные сообщества часто проводят конференции, в базе данных которых представлены такие сущности, как участники, статьи, залы для проведения заседаний, типы докладов организации и связи между ними.

В рамках этой задачи была создана база данных в Neo4j, которая позволяет легко исследовать конференцию различными типами категорий.

Она состоит из основных узлов: Company, Person и Presentation и связи: Is\_Sponsor\_Type, Works\_For, Presented\_By, Is\_Presentation\_Type, Is\_Category\_Type, Is\_CEU\_Type, Is\_In\_Room, Is\_In\_Session, Has\_Symbol. Через выбранные узлы (ноды) происходит связь с другими узлами: Sponsor Type, Room, Session, Symbol, Presentation Type, Category Type, CEU Type. Общая модель базы данных представлена на Рисунке 2.1.

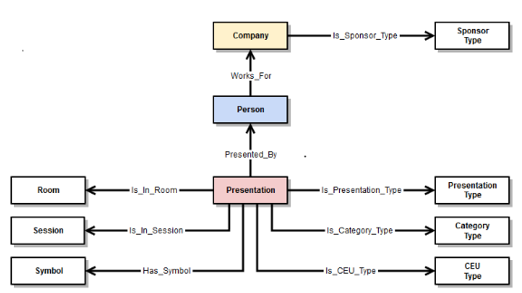


Рисунок 2.1 - Общая модель базы данных.

3. ВЫПОЛНЕНИЕ АНАЛИЗА ДАННЫХ

Язык Cypher позволяет строить различные по сложности запросы, сочетая в себе с одной стороны элементы SQL, а с другой графическое изображение связей.

На рисунке 3.1 изображена графовая база данных, которая отображает Общую модель базы данных с Рисунка 2.1.

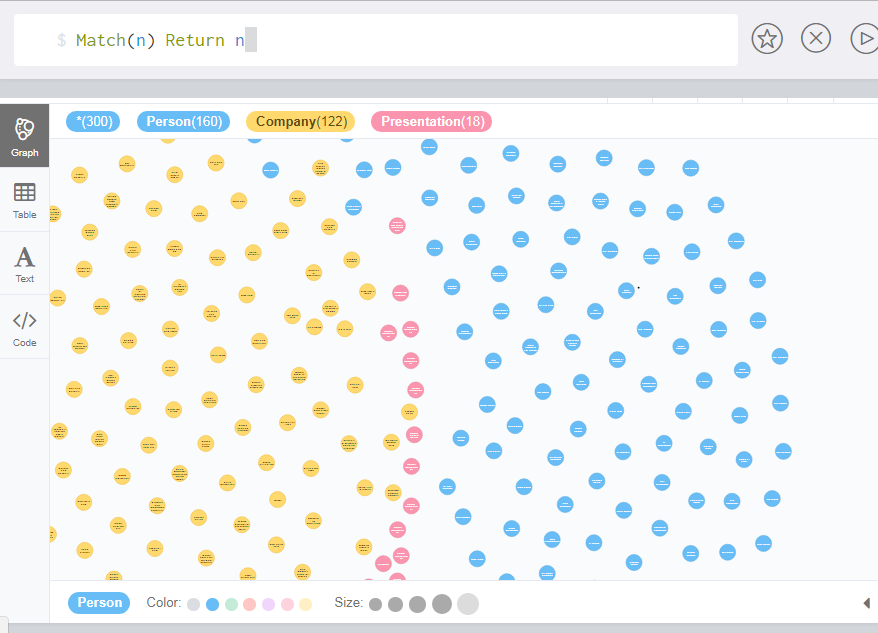


Рисунок 3.2 -

На данном изображении видны основные ноды БД, каждый тип из которых имеет свой цвет.

Для того, чтобы понимать, насколько важны связи в графовых СУБД, приведем пример, в котором иллюстрируется, как из связи, выбранной в левой части интерфейса, можно получить больше дополнительной информации, клкнув на нужный узел.

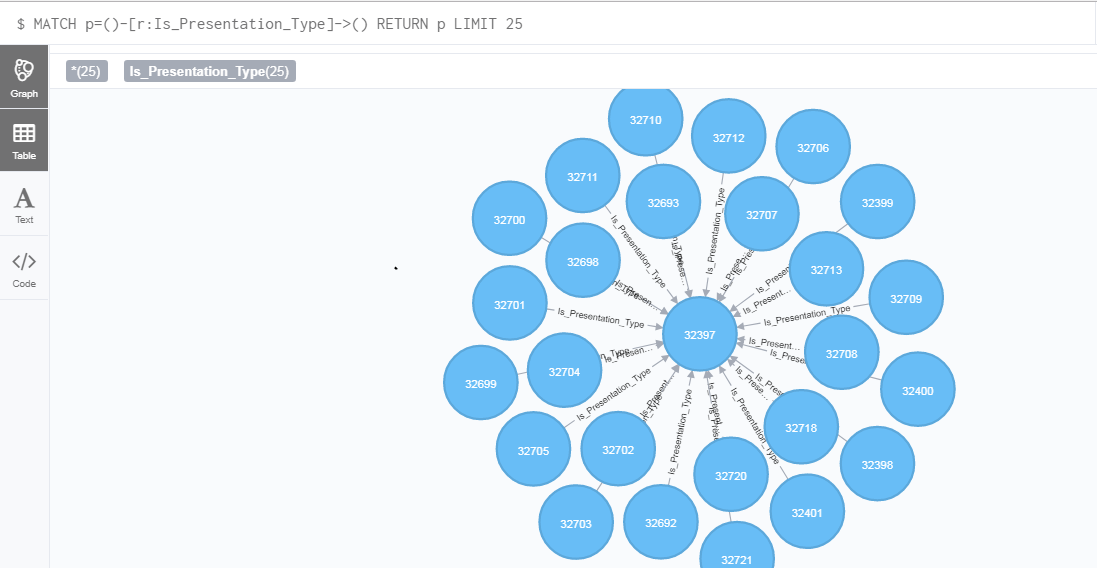


Рисунок 3.3 -

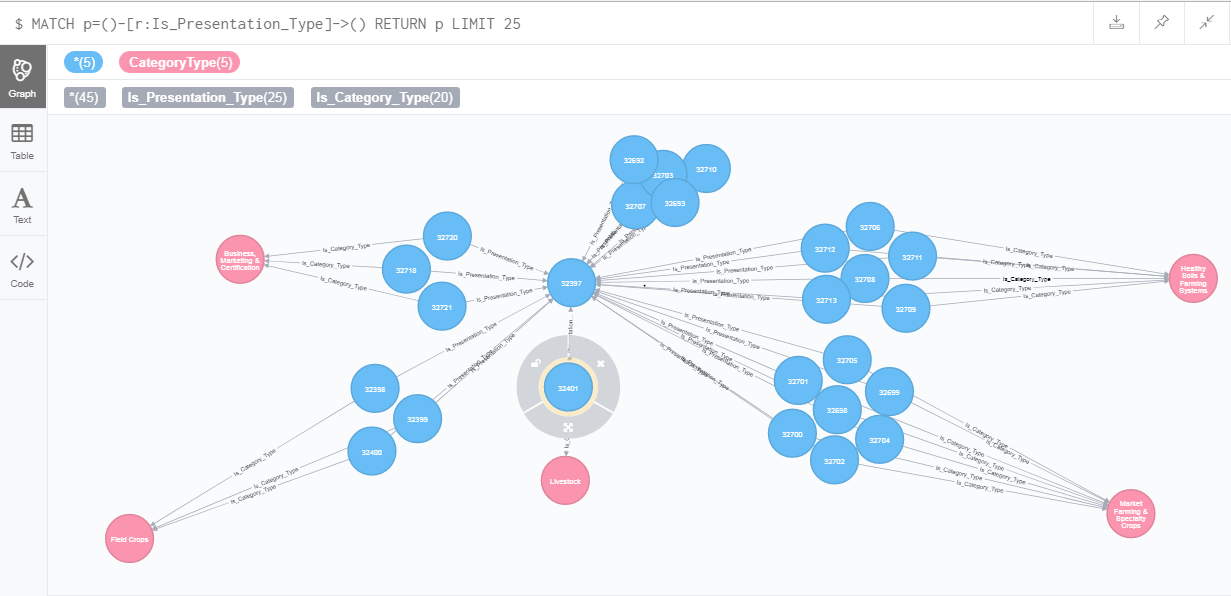


Рисунок 3.4 -

На Рисунке 3.3 был выбран тип связи Is\_Presentation\_Type. Как уже отмечалось выше, графовые базы данных позволяют получать вложенную информацию, что непосредственно иллюстрирует Рисунок 3.4.

Далее, приводятся более сложные запросы, с помощью которых можно извлечь необходимую информацию.

Таблица 3.1. Название

|  |  |
| --- | --- |
| Описание | Количество городов, представители которых участвуют в конференции |
| Запрос | MATCH (n:City) RETURN count(\*) |
| Ответ | |  | | --- | | count(\*) | | 39 | |

|  |  |
| --- | --- |
| Описание | Количество представленных презентаций в базе данных |
| Запрос | MATCH (n:Presentation) RETURN count(\*) |
| Ответ | |  | | --- | | count(\*) | | 137 | |

Как видно из приведенных выше таблиц, работа с простыми командами похожа на составление SQL-запросов. Следующий шаг – получение данных через описание связей. В этом случае и начинает работать «графическое изображение». В этом запросе данные можно представить, как в графическом виде, так и в табличном.

|  |  |
| --- | --- |
| Описание | Лица, у которых 3 и более презентаций на конференции. Будут выведены компании, в которых они работают. |
| Запрос1 (табличный вариант)  Ответ 1: | 1. MATCH (pr:Presentation)-[:Presented\_By]->(p:Person)   WITH p, COUNT(\*) AS Count  WHERE Count > 2  MATCH (p)-[Works\_For]->(c:Company)  WITH p, COLLECT(c) AS Companies, Count  RETURN p.name AS Name, Extract(c in Companies | c.name) AS `Company Names`, Count  ORDER BY Count DESC |
| Запрос 2 (графовый вариант) | 2)  MATCH (:Presentation)-[:Presented\_By]->(p:Person) WITH p, COUNT(\*) AS Count WHERE Count > 2 MATCH (pr:Presentation)-[Presented\_By]->(p)-[:Works\_For]->(c:Company) RETURN pr, p, c |
| Ответ 2: |  |

|  |  |
| --- | --- |
| Описание | 1. Необходимо узнать, присутствуют ли на конференции люди с вашего города, это могут быть продавцы, представители компаний или презентаторы. |
| Запрос | MATCH (c:Company)  WITH c.city AS NCity, c.state AS NState, COUNT(\*) AS Count  RETURN  CASE NCity  WHEN "" THEN "No City Listed"  ELSE NCity  END AS City,  CASE NState  WHEN "" THEN "No State Listed"  ELSE NState  END AS State, Count  ORDER BY Count DESC |
| Ответ |  |

|  |  |
| --- | --- |
| Описание | Вы хотите узнать, кто подготовил презентации на тему «Животноводство». |
| Запрос | MATCH (p:Presentation)-[:Is\_Category\_Type]->(c:CategoryType {name:'Livestock'}),  (p)-[:Is\_In\_Session]->(s:Session),  (p)-[:Is\_In\_Room]->(r:Room)  RETURN c.name AS CategoryType, p.name AS PresentationName, s.Name AS Session, r.Name AS Room |
| Ответ |  |

В работе была построена графовая база данных, которая имеет 1727 строк и представляет собой: 663 узла, 6213 свойств и 1064 отношения.

Используя средства языка Cypher, можно получить большое количество разной информации, в том числе и той, что показана в приведенных выше примерах. В запросах создавались узлы общей информации небольшого объема, но исходя из полученных данных, если грамотно построить базу данных, Neo4j дает возможность получить более глубокую информацию.

ЗАКЛЮЧЕНИЕ

Данная курсовая работа, направлена на исследование графовой базы данных Neo4j. Была разработана небольшая база данных на языке Cypher, на примере которой удалось провести больше количество экспериментов, тем самым разобраться в архитектуре и базовых возможностях выбранной СУБД, оценить ее интерфейс и SQL-подобный язык Cypher.

В процессе написания работы возникла проблема, связанная с соединением базы данных Neo4j с языком программирования Python. Изначально хотелось создавать базу данных в Python и там же прописывать все интересующие нас запросы, но т.к многочисленные тонкости подключения не удалось настроить, хотелось бы данную идею реализовать в следующей работе, и оперируя уже большими данными, ведь за свое существование Neo4j накопила большое количество графических алгоритмов, которые доступны в виде открытой библиотеки процедур, тем самым показать, в каких случаях графовые базы данных уместны в использовании, а когда –нет.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Ян Робинсон, Джим Вербер. Графовые базы данных.

|  |
| --- |
| 1. Onofrio Panzarino. Learning Cypher. 2. Sulit Gupta. Neo4j Essentials. |