Impact Factor:					
	ISRA (India) = 6.317	SIS (USA) = 0.912	ICV (Poland) = 6.630
	ISI (Dubai, UAE) = 1.582	РИНЦ (Russia) = 3.939	PIF (India) = 1.940
	GIF (Australia) = 0.564	ESJI (KZ) = 8.771	IBI (India) = 4.260
	JIF = 1.500	SJIF (Morocco) = 7.184	OAJI (USA) = 0.350

SOI: 1.1/TAS DOI: 10.15863/TAS

International Scientific Journal

Theoretical & Applied Science

p-ISSN: 2308-4944 (print) **e-ISSN:** 2409-0085 (online)

Year: 2022 **Issue:** 05 **Volume:** 109

Published: 25.05.2022 **http://T-Science.org**

Issue Article

Ekaterina Andreevna Matasova
Peter the Great St.Petersburg Polytechnic University
Bachelor's Student, Russian Federation
e-matasova@mail.ru

Oleg Yurievich Sabinin
Peter the Great St.Petersburg Polytechnic University
Associate Professor, candidate of technical sciences,
Russian Federation
olegsabinin@mail.ru

RESEARCH OF EFFICIENCY OF ORACLE AND NEO4J DBMS

Abstract: *The article discusses the advantages and disadvantages of a graph DBMS, and compares the capabilities of the Neo4j graph DBMS and the Oracle relational DBMS. As a result of the study, it was proved that there are tasks for which a graph data model is preferable to a relational one. The conducted research can serve as a basis for further study of the capabilities of graph and relational DBMS when executing various queries.*

Key words: *graph model, relational model, Neo4j DBMS, SQL, Cypher.*

Language: *Russian*

Citation: Matasova, E. A., & Sabinin, O. Yu. (2022). Research of efficiency of oracle and Neo4j DBMS. *ISJ Theoretical & Applied Science, 05 (109),* 742-752.

Soi: http://s-o-i.org/1.1/TAS-05-109-69 ***Doi***: https://dx.doi.org/10.15863/TAS.2022.05.109.69

Scopus ASCC: *1700.*

ИССЛЕДОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ РЕЛЯЦИОННОЙ СУБД ORACLE И ГРАФОВОЙ СУБД NEO4J

Аннотация: *В статье рассмотрены преимущества и недостатки графовых СУБД, а также проведено сравнение возможностей графовой СУБД Neo4j и реляционной СУБД Oracle. В результате исследования было доказано, что существуют задачи, для которых графовая модель данных предпочтительней реляционной. Проведённое исследование может служить основанием для дальнейшего изучения возможностей графовых и реляционных СУБД при выполнении различных запросов.*

Ключевые слова: *графовая модель, реляционная модель, СУБД Neo4j, язык запросов SQL, язык запросов Cypher.*

Введение

UDC 004.6

Наш мир активно развивается и объем данных, которые нужно хранить и обрабатывать ежедневно, растёт с небывалой быстротой. Учитывая количество генерируемой информации и необходимость ее анализа, становится понятно, что для ее обработки стандартные модели хранения могут быть далеко не самыми эффективными.

Обычно для хранения и обработки данных, содержащих сведения о некоторой предметной области, создается база данных. Самой распространенной моделью базы данных является реляционная модель. Однако, планомерный рост объемов данных в сети Интернет, заставил IT-сообщество задуматься над новыми стратегиями хранения и доступа к информации.

Данные часто подвергаются анализу и обработке, поэтому информация должна быть представлена в удобном для исследования виде.

Разработчики вынуждены пытаться каким-

либо образом усовершенствовать существующие решения или же искать новые пути к организации информации и их обработке.

Таким образом, всё более востребованными кажутся нестандартные подходы к хранению и анализу данных. И у каждого из этих способов есть свои преимущества и недостатки.

Обзор преимуществ использования графовых баз данных

База данных представляет собой совокупность специальным образом организованных данных, хранимых в памяти вычислительной системы и отображающих состояние объектов, и их взаимосвязей в рассматриваемой предметной области.

Данные, хранимые в базе, имеют определенную логическую структуру, которую называют моделью представления данных. Модель данных – средство абстракции, позволяющее видеть обобщённую структуру данных, хранимых в базе, а не их конкретные значения.

К основным моделям представления данных относятся следующие: иерархическая, сетевая, реляционная.

В настоящее время широкое распространение получила вариация сетевой модели – графовая модель данных.

Графовая модель данных на логическом уровне представляет собой направленный граф, состоящий из узлов и ребер. Узлы соответствуют объектам базы данных, а ребра – связям между этими объектами. Графовая модель хорошо отражает семантику предметной области с многочисленными связями. Например, пользователи социальной сети могут быть связаны между собой родственными, дружественными, производственными и прочими отношениями.

Одной причин выбора графовой модели данных является большой прирост производительности при работе со взаимосвязанными данными, которые удобно представить в форме графа.

Сегодня существует множество задач, которое реализуются с помощью графовых СУБД. Например, в медицинской сфере с помощью графовой модели представлены взаимосвязи между белками [1].

С помощью графовых систем созданы комплексы для анализа конкретных заболеваний. Например, существует система, моделирующая прогрессирование раковой опухоли. Она анализирует взаимосвязи между различными видами раковых опухолей, а также помогает отразить прогноз жизненного цикла опухоли с течением времени [2].

Кроме медицинской сферы, графовые технологии могут помочь при выборе косметики. Существует исследование, составляющее рейтинг безопасности различных косметических продуктов на основании данных о химических свойствах того или иного средства, информации о производителе, а также о количестве отозванной продукции [3].

Также графовые системы управления базами данных могут применяться и в различных областях IT. Почти вся активность пользователей, анализ геолокации и перемещений, а также мониторинг рекламных предпочтений клиента – для всего этого подходят графовые СУБД.

Преимущества использования графовой базы данных

Графовая база данных построена для работы с данными с большим количеством связей, а увеличение объема и связности данных предоставляет огромные возможности для устойчивого конкурентного преимущества.

Причины, по которым организации выбирают графовые базы данных [4; 5]:

- высокая производительность запросов;

Системы, работающие в онлайн режиме, должны отвечать на запросы пользователей очень быстро. В реляционных базах существенно снижается скорость выполнения запросов при росте объема информации и связей таблиц для поиска. Графовая база данных не использует индексный поиск, а преобразует операцию JOIN в обход графов, который происходит быстро и не зависит от общего размера информации.

- быстрое реагирование на изменение бизнес-логики;

Изменения в запросах пользователей приводят к изменению структурных требований. Графовая модель обладает большой гибкостью, так как существует возможность добавления новых видов взаимосвязей, новых узлов и новых подграфов в существующую структуру, не нарушая при этом существующих запросов и функционала.

- готовность внедрения в работу предприятия;

Многие графовые базы данных (в частности Neo4j) предоставляют продукт, соответствующий требованиям современных предприятий. Они поддерживают ACID свойства транзакций, обладают горизонтальной масштабируемостью, позволяют хранить миллионы сущностей, что может быть необходимо в крупных организациях.

Сравнение графовой и реляционной модели данных

Реляционный подход хранения данных

долгое время является основополагающим для создания различных систем хранения и управления информацией. Его в 1969 году представил Е.Ф. Кодд, известный исследователь в области баз данных, являвшийся на тот момент сотрудником фирмы IBM [6].

Реляционная база данных демонстрирует собой хранилище данных, которые организованы в виде таблиц. Эти таблицы состоят из строк или записей и столбцов (полей).

Данные в таблицах должны удовлетворять следующим показателям:

Значение, которое содержится в ячейке столбца и строки - атомарное;

Значения данных, которые находятся в одном столбце, относятся к одинаковому типу, доступному для использования в конкретной СУБД;

1. Все записи в таблице должны быть уникальны;
2. Поля имеют уникальные имена;
3. Порядок полей в структуре несущественен;
4. Порядок записей несущественен.

По структуре информация, хранящаяся в базе данных, представляет собой совокупность отношений, так как таблица, в которой каждый столбец A_i может содержать значения из множества $T_i = \{v_{i1}, v_{i2}, \ldots, v_{im}\}$ (все множества конечны), в математическом смысле представляет собой отношение над множествами $\{T_1, T_2, \ldots, T_n\}$.

Также в реляционной модели данных должна поддерживаться целостность, то есть отношения должны соответствовать стандартам целостности. Существуют несколько стандартов, такие как ограничение целостности сущности, или ограничение первичного ключа и ограничение ссылочной целостности, или ограничение внешнего ключа. Данные стандарты также должны поддерживаться в любой СУБД, реализующей реляционную модель.

Важно отметить, что реляционная модель включает в себя теорию нормализации, которая необходима для оптимизации построения базы данных и обеспечения более быстрого поиска информации [8].

Достоинством реляционной модели является простота, понятность и удобство физической реализации на ЭВМ, а также наличие строгого математического аппарата теории множеств, отношений и логики.

Реляционная СУБД, где данные структурированы в связанные таблицы, состоит из ограничений внешнего ключа и потому идеально подходит для обработки транзакций. Однако, если отношения между данными сложнее чем просто связи по ключу, а закономерности их связей неясны, реляционные СУБД становятся неэффективны.

В отличие от SQL-СУБД, графовые состоят из промаркированных вершин со свойствами, связанных между собой различными отношениями [10]. Отношения постоянно хранятся в базе данных, что ускоряет обход графа и сокращает время вычислений. Это позволяет находить новые закономерности в данных и отвечать на сложные вопросы. При этом совсем не нужно знать точный вопрос, если есть все данные и пути их соединения. Именно поэтому графовые базы данных отлично подходят для исследований связей между данными, например, поиск сообществ в соцсетях, обнаружения мошенничества и формирование рекомендаций.

Построение и сопровождение баз данных в Neo4j

Neo4j - графовая система управления базами данных, созданная на языке Java. Данные сохраняются в собственном формате, специально приспособленном под хранение информации в графовом виде, что позволяет использовать дополнительные средства оптимизации в случае данных со сложной структурой. Кроме этого, Neo4j имеет встроенную оптимизацию для работы с SSD-накопителями, при этом для обработки всего графа нет нужно целиком помещать его в оперативную память, что ощутимо ускоряет время выполнения различных запросов.

Важный аспект с точки зрения выполнения транзакций – поддержка принципа атомарности, согласованности, изолированности и устойчивости для каждой транзакции, что обеспечивает безопасность информации во время исполнения. Для работы с данной СУБД существует множество расширений. Например, есть возможность работы с Java, Clojure, Python и многими другими языками.

Ключевые компоненты Neo4j

Ключевые компоненты графовой базы данных - узлы, отношения и свойства(атрибуты).

Узлы — вершины графа, которые представляют объект или сущности.

Отношения — связи между любыми двумя узлами. Отношения имеют тип и направление. Типы обеспечивают общую категорию для каждого отношения. Отношения однонаправленны и специфичны: каждый узел может иметь много отношений с другими узлами, чтобы полностью описать контекст.

Свойства — конкретная информация каждого узла и отношения.

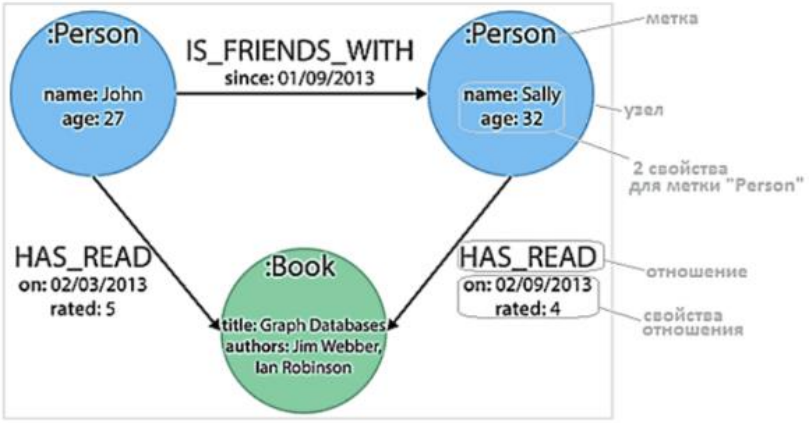

Рисунок 1 – Пример базы данных в СУБД Neo4j

Каждый узел в графовой модели базы данных содержит список записей, содержащих отношения с другими узлами. Всякий раз, когда пользователь выполняет эквивалент операции соединения в реляционной базе данных, графовая база данных использует этот список отношений, напрямую обращаясь к соединенным узлам, что избавляет от необходимости проводить дорогостоящие операции поиска и сопоставления.

Если значения некоторого свойства нет, то такое свойство будет отсутствовать на узле.

Таким образом, графовая модель позволяет описывать отношения более подробно, за счет именования отношений.

Особенности языка Cypher

Cypher — это язык запросов к графовой СУБД Neo4j, который позволяет вносить и извлекать данные из графа. В основе Cypher лежит SQL – язык для взаимодействия с реляционными базами. Тем не менее, Cypher был разработан и оптимизирован специально для работы с графовыми данными. Благодаря декларативности Cypher концентрируется на том, что выводится из базы, а не как эту информацию можно получить. Данная особенность важна, так как в случае изменении строения базы не будет необходимости полностью перестраивать все запросы, касающиеся измененной конфигурации. Это самый простой язык для работы с графовой моделью данных из-за его сходства с другими языками.

Cypher уникален тем, что обеспечивает визуальный способ сопоставления шаблонов и отношений. Cypher использует синтаксис типа ASCII-art, в котором используется формат:
(nodes)-[:ARE_CONNECTED_TO]->(otherNodes)

Таким образом, когда пишется запрос, вы рисуете рисунок графа через свои данные.

Графовая модель в Neo4j состоит из узлов и отношений, которые также могут иметь связанные с ними свойства. Однако узлы и отношения — это простые компоненты, которые создают наиболее мощную часть модели - шаблон. Шаблоны состоят из узлов и элементов отношений и могут выражать простые или сложные обходы в графе[10].

Cypher в значительной степени основан на шаблонах и предназначен для распознавания различных версий этих шаблонов, что делает его простым и логичным языком для изучения [11].

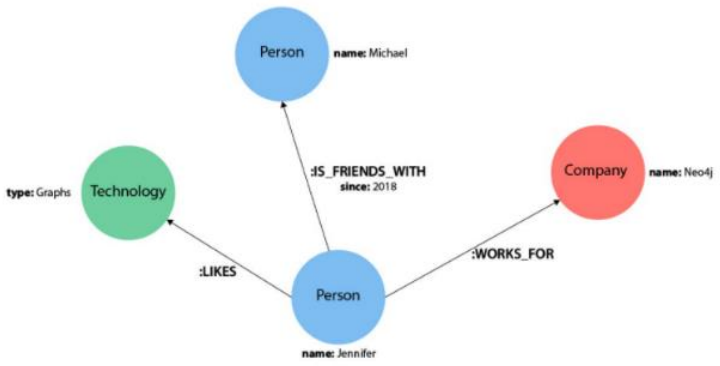

Рисунок 2 – Шаблон в СУБД Neo4j

Поскольку Cypher использует ASCII-Art для шаблонов, необходим визуальный способ представления каждого компонента шаблона выше. Основными компонентами модели графа являются узлы и отношения. Узлы — это объекты данных в графе. На картинке ниже Jennifer, Michael, Graphs и Neo4j — это узлы [11].

Для изображения узлов в Cypher используются круглые, например (node). Круглые скобки похожи на круги, которые визуальное представление использует для узлов в модели данных.

Если необходимо обратиться к узлу позже, можно присвоить ему переменную, например, (p) для человека или (t) для вещи. Обычно в реальных запросах используются более длинные и понятные имена переменных, например (person)или (thing). Как и в случае с другими языками программирования, можно называть переменные как угодно и ссылаться на них с тем же именем позже в запросе.

Если узел не имеет отношения к возвращаемым результатам, можно указать анонимный узел, используя пустые круглые скобки (). Это означает, что узел нельзя будет вернуть позже в запросе.

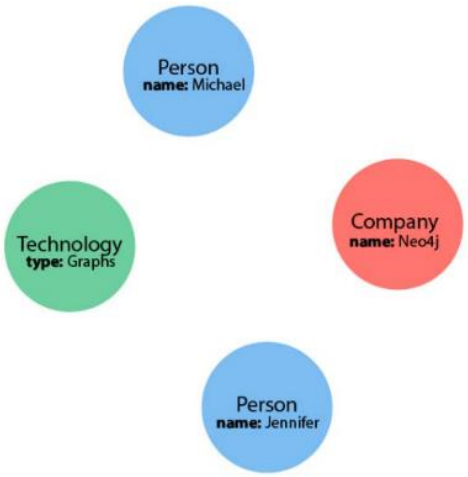

Рисунок 3 – Узлы в СУБД Neo4j

Чтобы полностью использовать возможности графовой базы данных, также необходимо выразить отношения между узлами. Отношения представлены в Cypher с помощью стрелки --> или <-- между двумя узлами. Дополнительная информация, например, как соединены узлы (тип отношения) и любые свойства, относящиеся к связи, могут быть помещены в квадратные скобки внутри стрелки.

На картинке ниже линии с LIKES, IS_FRIENDS_WITH и WORKS_FOR между узлами – это отношения.

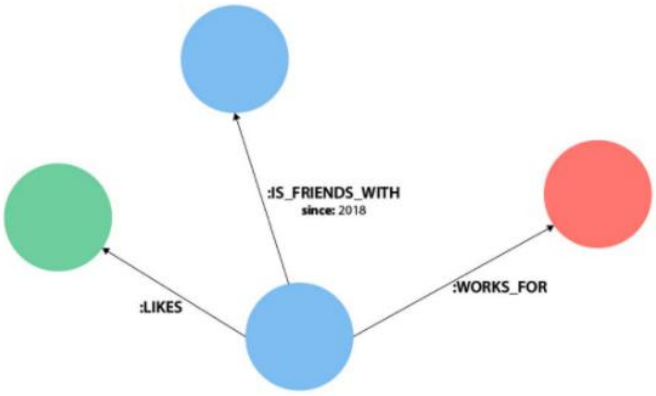

Рисунок 4 – Отношения в СУБД Neo4j

Ненаправленные отношения представлены без стрелки и только двумя черточками --. Это означает, что отношения могут развиваться в любом направлении. Хотя направление должно

быть в базе данных, оно может быть сопоставлено с ненаправленным отношением, при котором Cypher игнорирует какое-либо конкретное направление и извлекает взаимосвязь и подключенные узлы, независимо от физического направления. Это позволяет запросам быть гибкими и не заставлять пользователя знать физическое направление взаимосвязи, хранящейся в базе данных.

Если данные хранятся с одним направлением отношения, а запрос указывает неправильное направление, Cypher не вернет никаких результатов, поэтому, если направление неизвестно, лучше использовать ненаправленную связь.

Типы отношений классифицируют и добавляют значение отношениям. В графовой модели данных, представленной выше отношения, показывают, как узлы связаны друг с другом. Обычно отношения в модели данных можно определить по действиям или глаголам.

Можно указать любой тип отношений между узлами. Например, рассмотрим типы отношений из примера:

[:LIKES] - имеет смысл, когда узлы помещены по обе стороны от отношений (Jennifer LIKES Graphs)

[:IS_FRIENDS_WITH] - имеет смысл, когда мы помещаем в него узлы (Jennifer IS_FRIENDS_WITH Michael)

[:WORKS_FOR] - имеет смысл с узлами (Jennifer WORKS_FOR Neo4j)

Так же, как и с узлами, если необходимо обратиться к связи позже в запросе, можно присвоить ей переменную, например [rel]. Если не нужно ссылаться на отношения позже, можно указать анонимные отношения с помощью двух черточек --, -->, <--.

Последняя часть модели данных показывает свойства. Свойства — это пары имя-значение, которые предоставляют дополнительную информацию об узлах и отношениях. Чтобы представить их в Cypher, можно использовать фигурные скобки в круглых скобках узла или скобках отношения. Затем имя и значение свойства помещаются в фигурные скобки. В примере графа есть свойство узла (name) и свойство отношения (since).

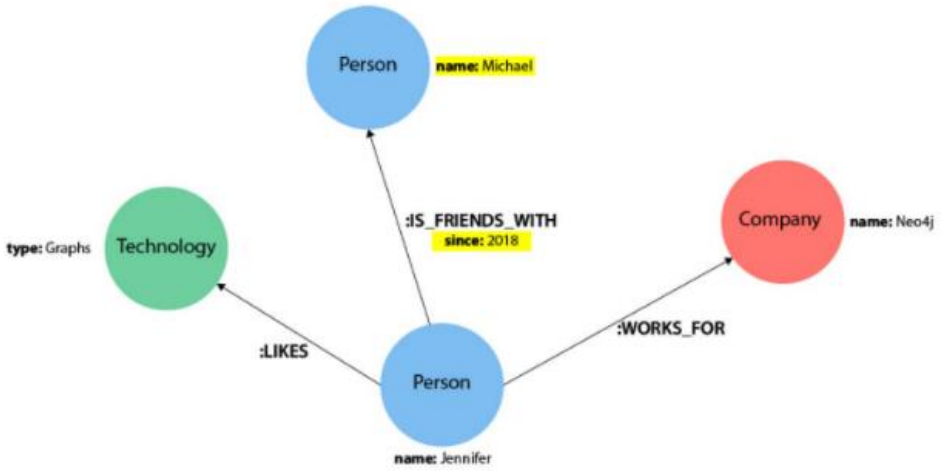

Рисунок 5 – Свойства в СУБД Neo4j

Свойство узла: (p:Person {name: 'Jennifer'})

Свойство отношения: -[rel:IS_FRIENDS_WITH {since: 2018}]->

Свойства могут иметь значения с различными типами данных.

Узлы и отношения составляют строительные блоки для графических паттернов. Эти строительные блоки могут объединяться в простые или сложные модели. Паттерны — это самая мощная возможность графов. В Cypher они могут быть записаны как непрерывный путь или разделены на более мелкие шаблоны и связаны запятыми.

Модель данных для исследования

Для того, чтобы проанализировать, какая из систем управления баз данных более эффективна с точки зрения обработки информации и скорости выполнения различных запросов, в первую очередь нужно создать модель того, как будет храниться информация в базе данных.

Чтобы использовать все плюсы системы управления графовыми базами данных, нужны данные, которые были бы сильно связанны. Иными словами, такая структура, где было бы максимальное количество связей между объектами. Один из подходящих под эти критерии набор данных – база данных

авиаперевозок.

Эта база данных будет использоваться работниками аэропортов и авиакомпаний.

Ниже приведена модель базы (рис. 6).

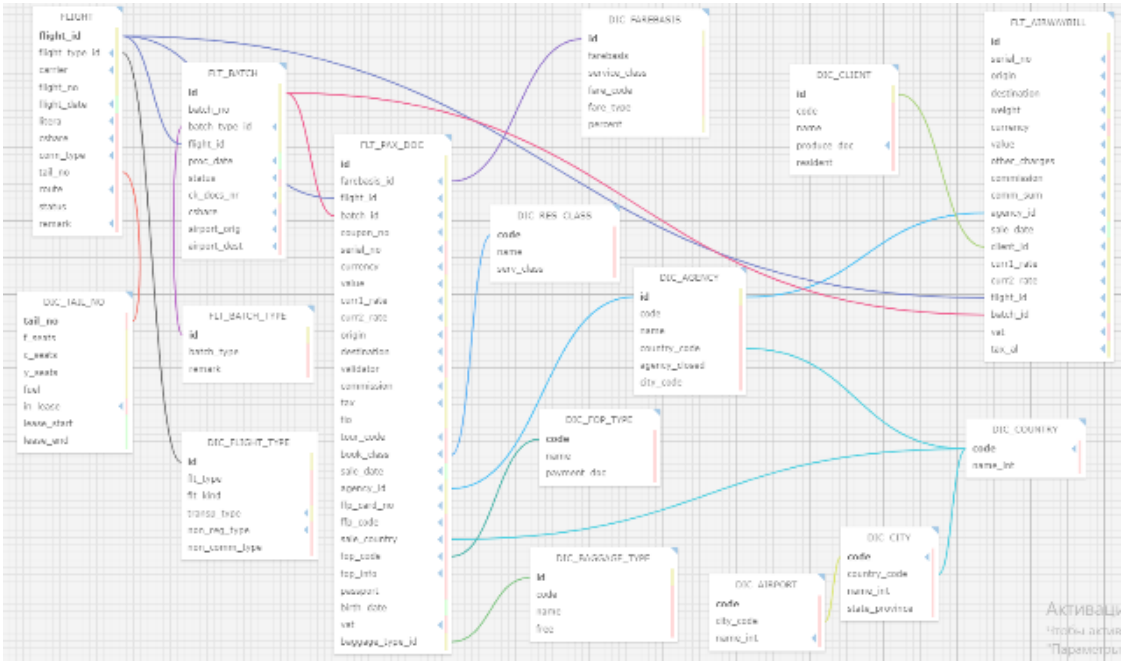

Рисунок 6 – Структура базы данных авиаперевозок

База данных, реализованная в Oracle, содержит таблицы:

- FLIGHT – таблица рейсов для отражения информации о рейсах
- FLT_BATCH – таблица пачек билетов, чтобы различать пассажиров, чартеры, груз
- FLT_PAX_DOC – таблица пассажирских документов для отражения информации о пассажире
- FLT_AIRWAYBILL – таблица грузовых накладных для отражения информации о грузах
- DIC_AGENCY - справочник агентов, которые продают билеты компании
- FLT_BATCH_TYPE – справочник типов пачек
- DIC_BAGGAGE_TYPE – справочник типов багажа
- DIC_RES_CLASS – справочник соответствия классов бронирования классам обслуживания (эконом, бизнес)
- DIC_CLIENT – справочник корпоративных клиентов
- DIC_FLIGHT_TYPE – справочник полетных типов
- DIC_FAREBASIS – справочник видов тарифа
- DIC_TAIL_NO – справочник бортов
- DIC_FOP_TYPE – справочник типов оплаты
- DIC_AIRPORT – справочник аэропортов
- DIC_CITY - справочник городов
- DIC_COUNTRY - справочник соответствия стран – городов

В графовой СУБД Neo4j для хранения данных будут использованы узлы и отношения.

Теперь каждый узел будет содержать информацию о рейсе, пассажире, грузовой накладной и т. д. Вместо полей таблицы, которые использовались в реляционной СУБД, в графовой будут использованы свойства.

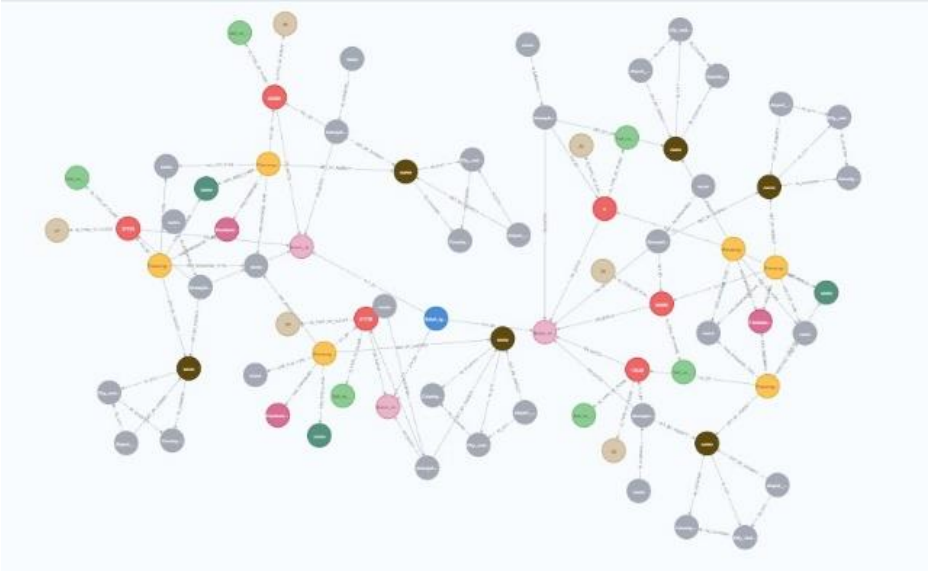

Рисунок 7 – Графовая модель базы данных авиаперевозок

Чтобы использовать все плюсы системы управления графовыми базами данных, нужен достаточно большой объем данных, чтобы разница по времени и эффективности была заметна.

Ниже приведена таблица с количеством записей в таблицах и количеством узлов в графовой базе (табл. 1).

Таблица 1. Объем информации в базе данных авиаперевозок

Имя таблицы	Количество строк в Oracle	Количество узлов в Neo4j
DIC_AGENCY	527	527
DIC_AIRPORT	2310	2310
DIC_BATCH_TYPE	37	37
DIC_BAGGAGE_TYPE	5	5
DIC_CITY	5428	5428
DIC_COUNTRY	234	234
DIC_FAREBASIS	4999	4999
DIC_FLIGHT_TYPE	30	30
DIC_FOP_TYPE	37	37
DIC_CLIENT	1000	1000
DIC_RES_CLASS	56	56
DIC_TAIL_NO	353	353
FLIGHT	5911	5911
FLT_AIRWAYBILL	12214	12214
FLT_BATCH	25358	25358
FLT_PAX_DOC	100000	100000

Исследование производительности реляционной и графовой СУБД

Чтобы точнее проанализировать работу различных систем управления баз данных, было решено создать несколько тестовых запросов различной сложности. Рассмотрим вариации тестов поподробнее.

Логично предположить, что графовый формат представления данных начнёт выигрывать (по времени выполнения) с увеличением количества связей в выполняемом запросе.

Рассмотрим несколько ситуаций, для тестирования каждой из которых произведено некоторое количество тестов (в среднем от 10 до 20 попыток, на каждый запрос).

Один из запросов, не содержащий соединений: топ-3 людей с максимальной стоимостью билета. Для этого необходимо из таблицы, в которой находится информация о пассажирах, выбрать ФИО пассажира, а также найти максимум от цены билета умноженную на

курс к рублю, чтобы стоимость была отражена в единой валюте:

```
select maximum, fio
from(
select max(value*CURR1_RATE)maximum , fio
from flt_pax_doc
group by fio
order by 1 desc)
where rownum<=3;
```

```
match(n:Passenger)
return n.fio as fio, max(n.value*n.curr1_rate) as max
order by max desc
limit 3
```

Рисунок 8 – Пример запроса без соединений в СУБД Oracle и Neo4

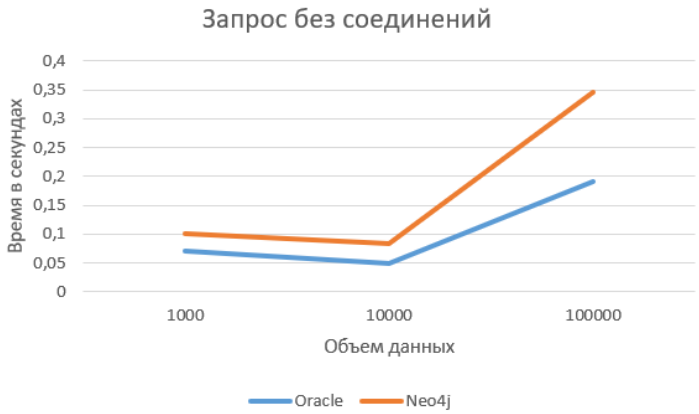

Рисунок 9 – Результат запроса без соединений в СУБД Oracle и Neo4j

Как видно из рисунка выше (рис. 9), при выполнении запроса без соединений реляционная база данных лучше справляется с обработкой запроса нежели графовая. Но ситуация в корне меняется при попытке выполнить запрос с большим количеством соединений. Например, нам необходимо для каждого пассажира получить информацию об агентстве, в котором был куплен билет. Для получения этих данных в реляционной СУБД необходимо соединить таблицы с информацией о пассажире, названии агентства, а также нужны таблицы стран и городов, чтобы показать местоположение агентства по продаже.

```
select px.fio, ag.code, ag.name, c.name_int, k.name_int
from
flt_pax_doc px
left join dic_agency ag on ag.id = px.agency_id
left join dic_city c on ag.city_code = c.code
left join dic_country k on k.code = c.country_code;
```

```
match (p:Passenger)-[bought_in_agency]-(a:Agency)-[is_in_city]-
(c:City)-[is_in_country]-(k:Country) return p.fio, a.name,
c.name_int, k.name_int
```

Рисунок 10 – Пример запроса с соединениями в СУБД Oracle и Neo4

Рисунок 11 – Результат запроса без соединений в СУБД Oracle и Neo4j

При анализе результатов, представленных на рисунке (рис. 11), можно сделать вывод о том, что даже при небольшом количестве соединений, хранение данных в виде графовой структуры более эффективно нежели реляционная модель.

Таким образом, исходя из проведенных тестов становится понятно, что эффективность использования графовых СУБД сильно зависит от модели анализируемых и хранимых данных. Данные использующиеся при поведенческом анализе действий пользователя, большое количество информации, применяемое для исследования медицинских показателей, информация о перевозках и многое другое, что имеет нативную графовую структуру лучше всего подходят для обработки в графовых системах.

Что же касается данных, для которых не характерна графовая структура, например, бухгалтерская информация или личные данные пользователя социальной сети – всё это плохо сочетается с графовой моделью. Но чем больше количество связей внутри подобной структуры – тем меньше будет отставание графовой СУБД от реляционной.

Выводы

Таким образом, сегодня развитие бизнеса привело не только к значительному увеличению объемов данных, но и существенному их усложнению, и реляционные СУБД просто перестали справляться с таким объемам информации. Разочарование в реляционной модели привело к появлению альтернативных моделей, учитывающих новые реалии. Стали появляться и активно развиваться альтернативные способы хранения информации, получившие название NoSQL.

Базы данных NoSQL предназначены для работы с целым рядом шаблонов доступа к данным. Наибольший интерес сегодня представляет графовая модель данных.

Во-первых, графовая модель сама по себе является наиболее естественным подходом к моделированию. Во-вторых, наше время характеризуется ростом связности данных. Самый яркий тому пример — бум социальных сетей, на логическом уровне представляющих собой большие сильно связанные графы. Наконец, укрупнение и интеграция информационных сервисов, характерные для современного этапа развития корпоративных ИТ, сопровождаются объединением разнородных баз данных в единые комплексы, что весьма затратно реализуется на основе реляционной модели, но практически безболезненно может быть произведено с использованием графовой.

References:

1. Lysenko, A., et al. (n.d.). *Representing and querying disease networks using graph databases*. Retrieved 20.05.2022 from https://biodatamining.biomedcentral.com/articles/10.1186/s13040-016-0102-8
2. Johnson, D. (n.d.). *Semantically Linking Cancer Models.* Retrieved 20.05.2022 from https://journals.sagepub.com/doi/10.4137/CIN.S13895

3. Chau, N. Z. (n.d.). *Chemicals In Cosmetics*. Retrieved from https://portal.graphgist.org/graph_gists/chemicals-in-cosmetics
4. (n.d.). *Model: The Definitive Guide to Graph Databases.* Retrieved 20.05.2022 from https://neo4j.com/whitepapers/rdbms-developers-graph-databases-ebook/
5. Watt, A., Abedrabbo, T., & Fox, D. (2015). *Neo4j in Action.* (p.268). Manning Publications Co..
6. Kuznetsov, S. (n.d.). *Database. Introductory course.* Retrieved 20.05.2022 from http://citforum.ru/database/advanced_intro
7. (n.d.). *Comparing SQL with Cypher*. Retrieved 20.05.2022 from https://neo4j.com/developer/cypher/guide-sql-to-cypher/
8. Sergeeva, T.I., & Sergeev, M.Yu. (n.d.). *Databases: data models, design, SQL language*. (p.233). Voronezh: FGBOY HPO "Voronezh State Technical University". Retrieved 20.05.2022 from https://cchgeu.ru/upload/iblock/04d/metod_rkis_ivt_ras_24.06.2016.pdf
9. (n.d.). *Glossary of keywords.* Retrieved 20.05.2022 from https://neo4j.com/docs/cypher-manual/current/keyword-glossary/
10. Vukotic, A., & Watt, N. (2015). *Learning Neo4j*. (p.214). Packt Publishing.
11. (n.d.). *Getting Started with Cypher*. Retrieved 20.05.2022 from https://neo4j.com/developer/cypher/intro-cypher/
12. Novik, V.I., & Sabinin, O.Yu. (2019). *A control system based on a graph database*. SPBPU Science Week. (pp. 138-141).