**ЛАБОРАТОРНАЯ РАБОТА №8**

**ИССЛЕДОВАНИЕ ПРИНЦИПОВ РАБОТЫ С ГРАФОВЫМИ БД**

**8.1 Цель работы**

Рассмотреть основные принципы моделирования структуры данных с использованием графовой модели, познакомиться с наиболее популярными сферами деятельности, где возможно более эффективное её применение по сравнению с реляционной моделью, выполнить настройку программной среды для работы с СУБД Neo4j.

**8.2 Порядок выполнения работы**

8.2.1 Выполнить инсталляцию сервера базы данных Neo4j и инструменты разработчика, создать новый проект и базу данных.

8.2.2 Разработать базу данных для предметной области по вариантам. Не менее 5 типов узлов и 5 типов связей. Написать код, генерирующий узлы и связи между узлами.

8.2.3 Используя полученную базу данных, выполнить и описать результаты запросов, представленных в рамках данной лабораторной работы (поиск, сортировка, фильтрация, изменение данных, удаление данных, использование агрегатных функций).

8.2.4 Измерить скорость выполнения запросов для следующего количества узлов: 100, 1000, 10 000.

8.2.5 Оформить отчет по работе.

**8.3 Индивидуальный вариант №3**

Предметная область: Социальная сеть.

**8.4 Ход работы**

8.4.1 Был установлен Neo4j Desktop, создано новое подключение и база данных (рисунок 8.1).

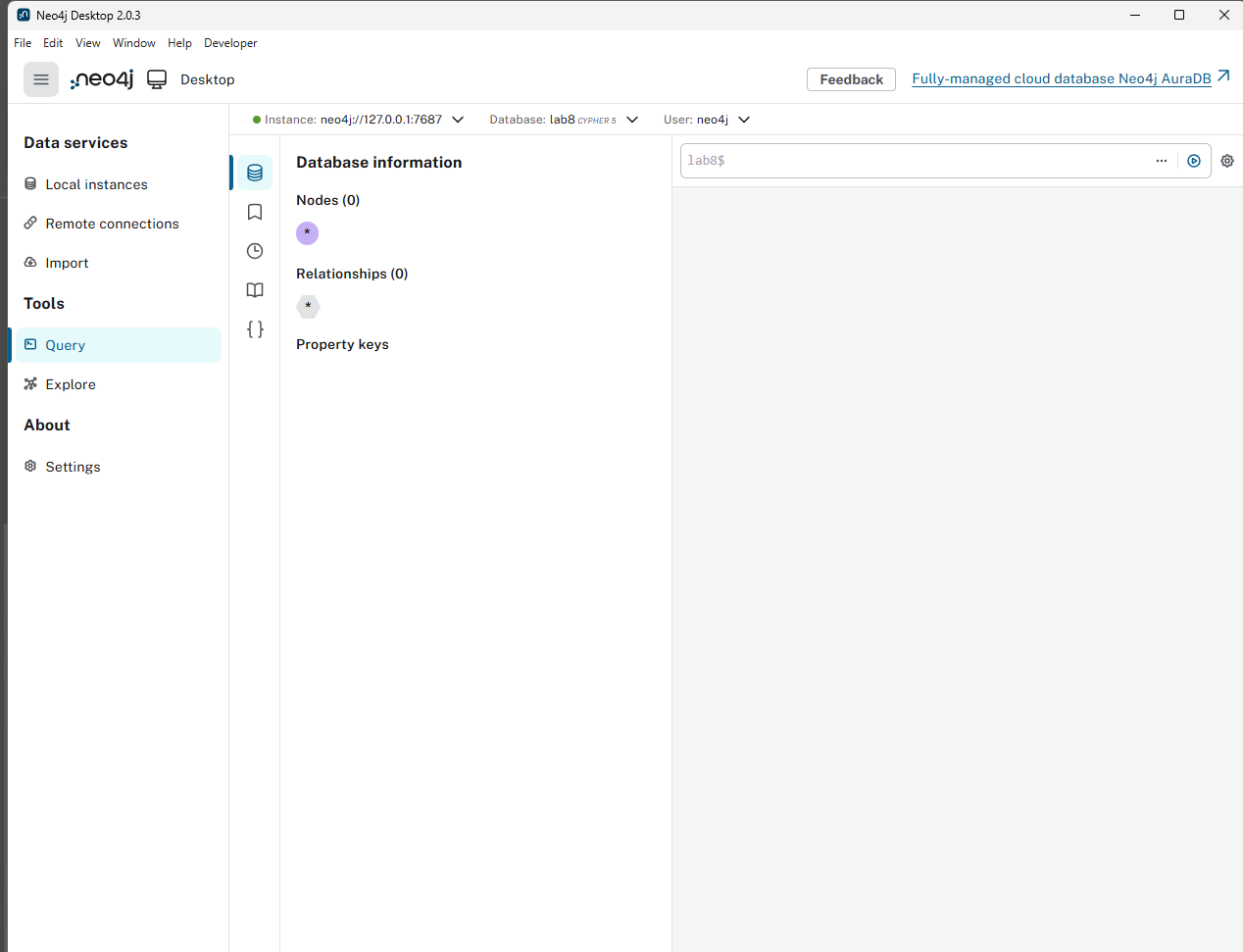


Рисунок 8.1 – Окно Neo4j Desktop

8.4.2 Были определены типы узлов и связей для базы данных. Типы были отражены в таблице (таблицы 8.1 – 8.2)

Таблица 8.1 – Типы узлов

|  |  |  |
| --- | --- | --- |
| Тип узла | Описание | Свойства |
| User | Пользователь | Id, name, email, registered\_at, avatar\_url |
| Post | Пост в соц. сети | Id, content, created\_at, media\_url |
| Comment | Комментарий, оставленный пользователем | Id, text, created\_at |
| Group | Сообщество | Id, name, created\_at, desctiption, logo\_url |
| Story | История | Id, created\_at, content\_url, expires\_at |

Таблица 8.2 – Типы связей

|  |  |
| --- | --- |
| Типы связей | Описание |
| FOLLOWS | Пользователь подписан на другого пользователя |
| FRIEND | Пользователи друзья |
| LIKED | Пользователь лайкнул пост, комментарий или историю |
| CREATED | Пользователь опубликовал пост, комментарий или историю |
| MEMBER\_OF | Пользователь состоит в сообществе |
| COMMENTED\_ON | Пользователь оставил комментарий под постом |

8.4.3 Был написан код для генерации тестовых данных (листинги 8.1-).

Листинг 8.1 – Код запросов для генерации пользователей

WITH range(1, 100) AS ids

UNWIND ids AS id

CREATE (:User {

id: id,

name: "User\_" + id,

email: "user\_" + id + "@example.com",

registered\_at: date() - duration({days: id}),

avatar\_url: "avatar\_" + id + ".png"

});

В листинге 8.1 генерируются 100 записей, для генерации большего количества узлов, число заменяется на соответствующее.

Листинг 8.2 – Код запроса для генерации постов и связей с пользователями

CALL apoc.periodic.iterate(

"MATCH (u:User) RETURN u",

"WITH u, rand() AS r

WHERE r < 0.7

WITH u, toInteger(rand() \* 5) AS postCount

UNWIND range(1, postCount) AS i

CREATE (p:Post {

id: u.id \* 10 + i,

content: 'Post #' + u.id + '\_' + i,

created\_at: datetime() - duration({minutes: i}),

media\_url: 'media\_' + u.id + '\_' + i + '.jpg'

}),

(u)-[:CREATED]->(p)",

{batchSize: 500, parallel: false}

)

Здесь учтено, что не все пользователи публикуют посты (для примера взяли 70%). Также у одного пользователя может быть несколько постов, поэтому, при генерации, на каждого пользователя из 70% может быть от 1 до 4 постов.

Листинг 8.3 – Код запроса для генерации комментариев к постам

CALL apoc.periodic.iterate(

"MATCH (p:Post) RETURN p",

"WITH p, toInteger(rand() \* 4) AS commentCount

UNWIND range(1, commentCount) AS i

MATCH (u:User)

WITH p, u, i, rand() AS r

WHERE r < 0.05

CREATE (c:Comment {

id: p.id \* 100 + i,

text: 'Comment on post ' + p.id + ' by user ' + u.id,

created\_at: datetime() - duration({minutes: i})

}),

(u)-[:CREATED]->(c),

(c)-[:COMMENTED\_ON]->(p)",

{batchSize: 100, parallel: false} //100 постов

)

Здесь для каждого поста совершается по 3 попытки на каждого пользователя оставить комментарий. Шанс, что пользователь оставит комментарий 5%. Таким образом, от каждого пользователя на каждом посте может быть от 0 до 3 комментариев. Шанс того, что комментарии будут под каждым постом в данном запросе достаточно низок.

Листинг 8.4 – Код запроса для генерации историй и связей с пользователями

CALL apoc.periodic.iterate(

"MATCH (u:User) RETURN u",

"WITH u, rand() AS r

WHERE r < 0.3

WITH u, toInteger(rand() \* 4) AS storyCount

UNWIND range(1, storyCount) AS i

CREATE (s:Story {

id: u.id \* 10 + i,

created\_at: datetime() - duration({hours: i}),

content\_url: 'story\_' + (u.id \* 10 + i) + '.mp4',

expires\_at: datetime() + duration({hours: 24})

}),

(u)-[:CREATED]->(s)",

{batchSize: 100, parallel: false}

)

Способ генерации аналогичен генерации постов.

Листинг 8.5 – Код запроса для генерации сообществ и связей с пользователями

WITH range(1, 40) AS groupIds

UNWIND groupIds AS groupId

CREATE (:Group {

id: groupId,

name: "Group\_" + groupId,

created\_at: date() - duration({days: groupId}),

description: "Описание сообщества #" + groupId,

logo\_url: "group\_" + groupId + ".png"

});

CALL apoc.periodic.iterate(

"MATCH (u:User) RETURN u.id AS uid",

"MATCH (u:User {id: uid})

MATCH (g:Group)

WITH u, g, rand() AS r

WHERE r < 0.1

CREATE (u)-[:MEMBER\_OF]->(g)",

{batchSize: 100, parallel: false}

);

Листинг 8.6 – Код запроса для генерации постов сообществ

CALL apoc.periodic.iterate(

"MATCH (g:Group) RETURN g",

"WITH g, g.id AS groupId, rand() AS r

WHERE r < 0.8

WITH g, groupId, toInteger(rand() \* 7) + 1 AS postCount

UNWIND range(1, postCount) AS i

CREATE (p:Post {

id: groupId \* 1000 + i,

content: 'Group Post #' + groupId + '\_' + i,

created\_at: datetime() - duration({hours: i}),

media\_url: 'group\_media\_' + groupId + '\_' + i + '.jpg'

}),

(g)-[:CREATED]->(p)",

{batchSize: 500, parallel: false}

)

Способ генерации постов сообществ аналогичен генерации постов пользователей. На посты сообществ пользователи также могут оставлять комментарии.

Листинг 8.7 – Код запроса для генерации подписчиков и друзей пользователя

CALL apoc.periodic.iterate(

"MATCH (u1:User), (u2:User) WHERE u1.id < u2.id RETURN u1.id AS id1, u2.id AS id2",

"MATCH (u1:User {id: id1}), (u2:User {id: id2})

WITH u1, u2, rand() AS r

WITH u1, u2,

CASE

WHEN r < 0.2 THEN 'FRIEND'

WHEN r < 0.6 THEN 'FOLLOWS'

ELSE 'NONE'

END AS relType

FOREACH (\_ IN CASE WHEN relType = 'FRIEND' THEN [1] ELSE [] END |

CREATE (u1)-[:FRIEND]->(u2),

(u2)-[:FRIEND]->(u1)

)

FOREACH (\_ IN CASE WHEN relType = 'FOLLOWS' THEN [1] ELSE [] END |

CREATE (u1)-[:FOLLOWS]->(u2)

)",

{batchSize: 100, parallel: false}

)

В данном запросе 40% пользователей подписаны на кого-то, а 20% дружат.

Листинг 8.8 – Код запроса для генерации лайков на истории, посты или комментарии

CALL apoc.periodic.iterate(

"MATCH (u:User) RETURN u.id AS uid",

"MATCH (u:User {id: uid})

MATCH (p:Post)

WHERE NOT (u)-[:CREATED]->(p)

WITH u, p, rand() AS r

WHERE r < 0.05

CREATE (u)-[:LIKED]->(p)",

{batchSize: 100, parallel: false}

);

CALL apoc.periodic.iterate(

"MATCH (u:User) RETURN u.id AS uid",

"MATCH (u:User {id: uid})

MATCH (c:Comment)

WHERE NOT (u)-[:CREATED]->(c)

WITH u, c, rand() AS r

WHERE r < 0.05

CREATE (u)-[:LIKED]->(c)",

{batchSize: 100, parallel: false}

);

CALL apoc.periodic.iterate(

"MATCH (u:User) RETURN u.id AS uid",

"MATCH (u:User {id: uid})

MATCH (s:Story)

WHERE NOT (u)-[:CREATED]->(s)

WITH u, s, rand() AS r

WHERE r < 0.05

CREATE (u)-[:LIKED]->(s)",

{batchSize: 100, parallel: false}

);

В запросах, начиная с листинга 8.2 была использована процедура «apoc.periodic.iterate» плагина APOC. Данная процедура предназначена для пакетной обработки больших объёмов данных. Без использования данной процедуры, при больших объёмах данных, запросы могут вызывать переполнение памяти. Процедура позволяет разбить запрос, который обрабатывает большой объём данных на несколько запросов, которые обрабатывают меньший набор данных. Например, при batchSize: 100 в одной итерации будет обработано 100 элементов из набора данных. Набор данных определяется 1м аргументом, во 2м аргументе сам запрос, а в 3м аргументе указываются дополнительные параметры.

8.4.4 Были выполнены запросы для 100, 1000 и 10000 узлов. Было зафиксировано время выполнения при различном количестве записей. Здесь количество узлов относится именно к количеству пользователей, количество остальных узлов определяется динамически на основании количества пользователей.

Были выполнены несколько запросов поиска для разного количества записей.

1) Найти всех пользователей, которые ведут активно ведут социальную сеть (публикуют посты, истории и комментарии). Код запроса представлен в листинге 8.9.

Листинг 8.9 – Код запроса поиска активных пользователей

MATCH (u:User)-[:CREATED]->(:Story)

MATCH (u)-[:CREATED]->(:Post)

MATCH (u)-[:CREATED]->(:Comment)

RETURN u;

Пример результата представлен на рисунке 8.2.

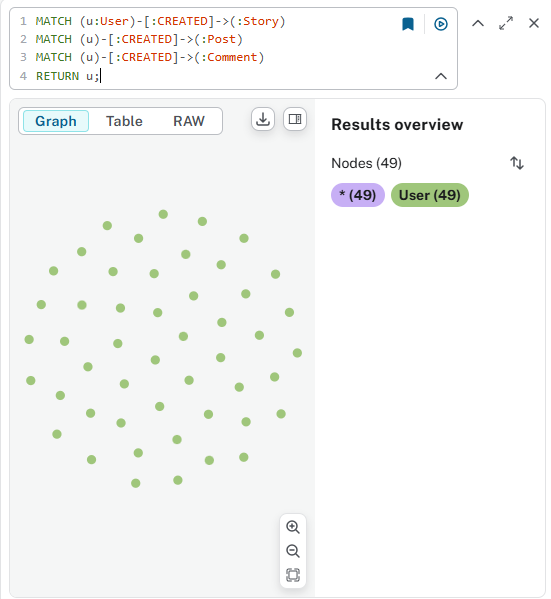


Рисунок 8.2 – Пример выполнения запроса 1

2) Найти пользователей, которые состоят в сообществах. Код запроса представлен в листинге 8.10.

Листинг 8.10

MATCH (u:User)-[:MEMBER\_OF]->(:Group)

RETURN u;

Пример результата выполнения запроса представлен на рисунке 8.3.

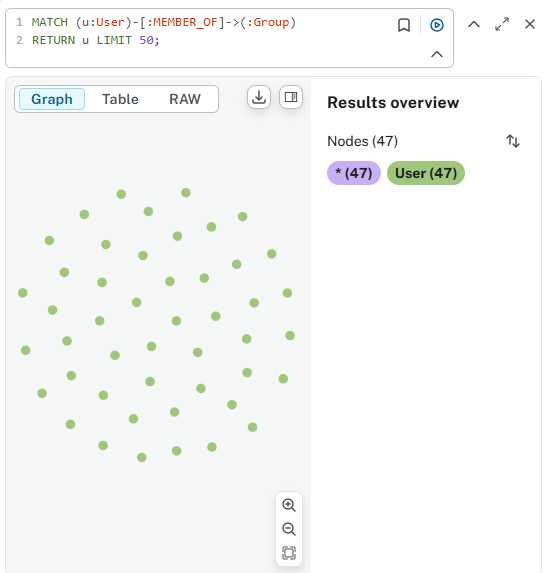


Рисунок 8.3 – Пример выполнения запроса 2

Были выполнены запросы поиска с использованием сортировки.

3) Найти всех пользователей, которые ведут активно ведут социальную сеть (публикуют посты, истории и комментарии). Полученные данные отсортировать по дате регистрации в порядке убывания. Код запроса представлен в листинге 8.11.

Листинг 8.11 – Код запроса поиска активных пользователей с использованием сортировки

MATCH (u:User)-[:CREATED]->(:Post)

MATCH (u)-[:CREATED]->(:Story)

MATCH (u)-[:CREATED]->(:Comment)

RETURN u

ORDER BY u.registered\_at DESC;

Пример результата выполнения запроса представлен на рисунке 8.4.

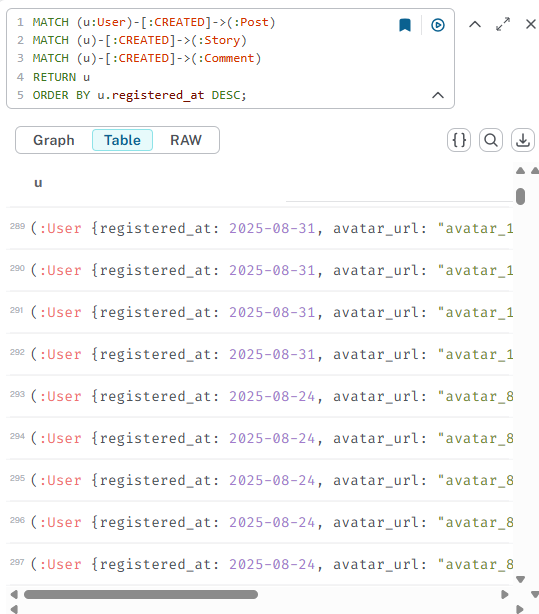


Рисунок 8.4 – Пример выполнения запроса 3

Были выполнены запросы поиска с применением фильтрации.

4) Найти всех друзей друзей пользователя с id 1. Полученные данные отсортировать в порядке убывания имени пользователя. Код запроса представлен в листинге 8.12.

Листинг 8.12 – Код запроса поиска пользователей без друзей и подписчиков

MATCH (u:User {id: 1})-[:FRIEND]-(friend:User)-[:FRIEND]-(friendOfFriend:User)

WHERE friendOfFriend.id <> u.id AND NOT (u)-[:FRIEND]-(friendOfFriend)

RETURN DISTINCT friendOfFriend

ORDER BY friendOfFriend.name DESC

Пример результата выполнения запроса представлен на рисунке 8.5.

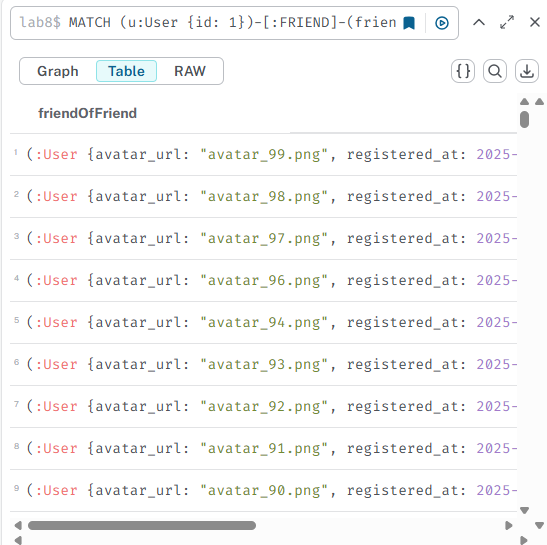


Рисунок 8.5 – Пример выполнения запроса 4

5) Найти пользователей, которые пишут комментарии под постами, но при этом не состоят ни в одном из сообществ. Полученные данные отсортировать в порядке убывания имени пользователя. Код запроса представлен в листинге 8.13.

Листинг 8.13 – Код запроса

MATCH (u:User)-[:CREATED]->(c:Comment)-[:COMMENTED\_ON]->(:Post)

WHERE NOT (u)-[:MEMBER\_OF]->(:Group)

RETURN DISTINCT u.name AS userName

ORDER BY userName DESC;

Пример результата выполнения запроса представлен на рисунке 8.6.

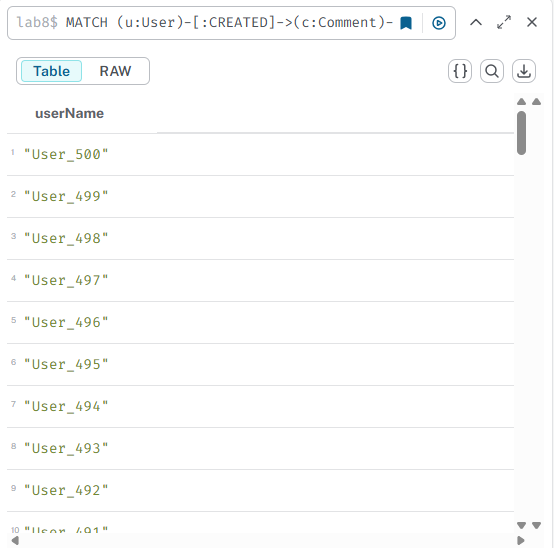


Рисунок 8.6 – Пример выполнения запроса 5

6) Найти посты, под которыми нет комментариев и не стоит хотя бы один лайк. Код запроса представлен в листинге 8.14.

Листинг 8.14 – Код запроса

MATCH (p:Post)

WHERE NOT (p)<-[:COMMENTED\_ON]-(:Comment)

AND NOT (p)<-[:LIKED]-(:User)

RETURN p;

Пример результата выполнения запроса представлен на рисунке 8.7.

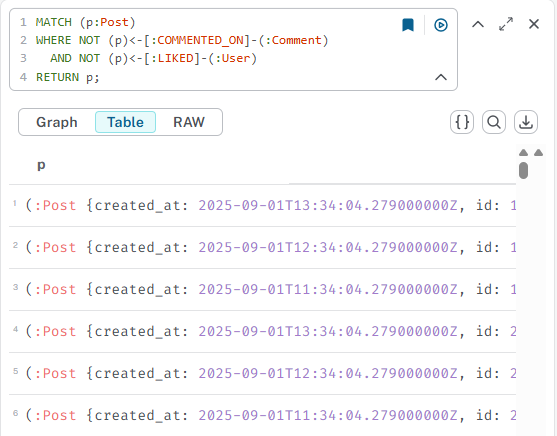


Рисунок 8.7 – Пример выполнения запроса 6

Были выполнены запросы с изменением данных.

7) Изменить аватар User\_54 на new\_avatar.png. Код запроса представлен в листинге 8.15.

Листинг 8.15 – Код запроса на изменение данных

MATCH (u:User {name: "User\_54"})

SET u.avatar\_url = "new\_avatar.png";

Пример результата выполнения запроса представлен на рисунке 8.8

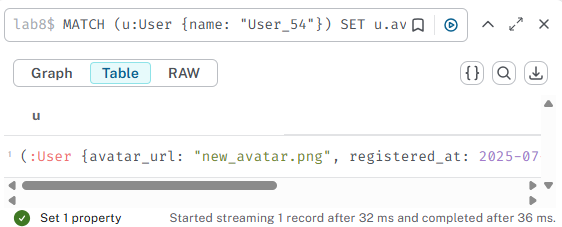


Рисунок 8.8 – Пример выполнения запроса 7

Были выполнены запросы на удаление данных.

8) Удалить все истории, опубликованные более суток назад. Код запроса представлен в листинге 8.16.

Листинг 8.16 – Код запроса удаления данных

MATCH (s:Story)

WHERE s.expires\_at < datetime()

DETACH DELETE s;

Пример результата выполнения запроса представлен на рисунке 8.9.

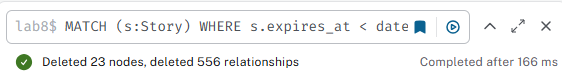


Рисунок 8.9 – Пример выполнения запроса 8

Были выполнены запросы с использованием агрегатных функций.

9) Найти 10 самых активных пользователей. Отобразись количество постов, комментариев, лайков, историй. Отсортировать по активности. Код запроса представлен в листинге 8.17.

Листинг 8.17 – Код запроса поиска 10 самых активных пользователей

MATCH (u:User)

OPTIONAL MATCH (u)-[:CREATED]->(p:Post)

WITH u, count(p) AS postCount

OPTIONAL MATCH (u)-[:CREATED]->(c:Comment)

WITH u, postCount, count(c) AS commentCount

OPTIONAL MATCH (u)-[:CREATED]->(s:Story)

WITH u, postCount, commentCount, count(s) AS storyCount

OPTIONAL MATCH (u)-[:LIKED]->(lp:Post)

WITH u, postCount, commentCount, storyCount, count(lp) AS postLikes

OPTIONAL MATCH (u)-[:LIKED]->(lc:Comment)

WITH u, postCount, commentCount, storyCount, postLikes, count(lc) AS commentLikes

OPTIONAL MATCH (u)-[:LIKED]->(ls:Story)

WITH u, postCount, commentCount, storyCount, postLikes, commentLikes, count(ls) AS storyLikes

WITH u.name AS userName,

postCount,

commentCount,

storyCount,

(postLikes + commentLikes + storyLikes) AS likeCount,

(postCount + commentCount + storyCount + postLikes + commentLikes + storyLikes) AS totalActivity

RETURN userName, postCount, commentCount, storyCount, likeCount, totalActivity

ORDER BY totalActivity DESC

LIMIT 10;

Пример результата выполнения запроса представлен на рисунке 8.10.

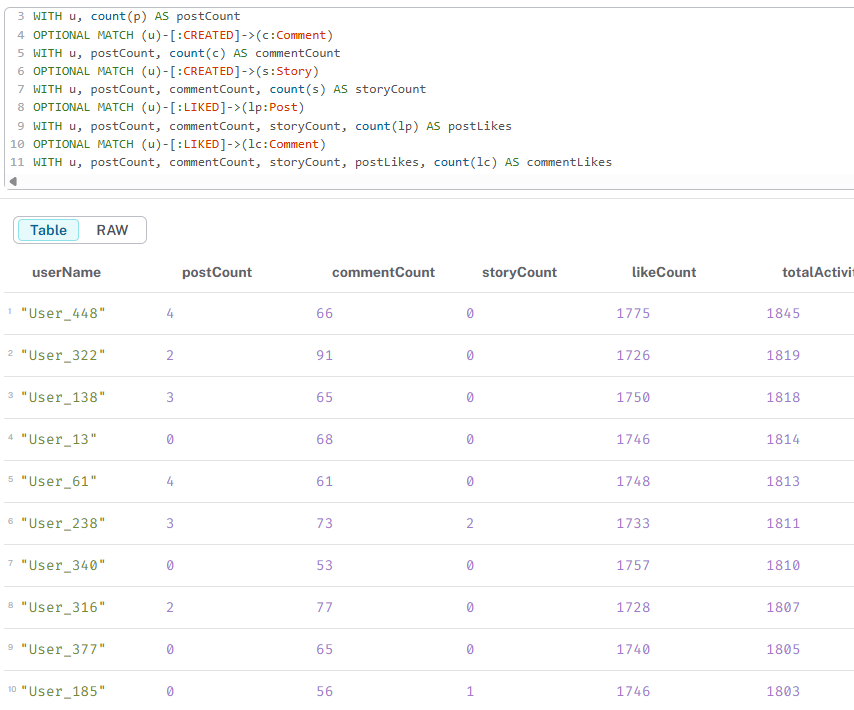


Рисунок 8.10 – Пример выполнения запроса 9

10) Найти 10 пользователей, которые получили больше всех лайков. Отсортировать по количеству лайков. Код запроса представлен в листинге 8.18.

Листинг 8.18 – Код запроса

MATCH (u:User)-[:CREATED]->(c)

WHERE c:Post OR c:Comment OR c:Story

OPTIONAL MATCH (:User)-[:LIKED]->(c)

WITH u, count(\*) AS likeCount

RETURN u.name AS userName, likeCount

ORDER BY likeCount DESC

LIMIT 10;

Пример результата выполнения запроса представлен на рисунке 8.11.

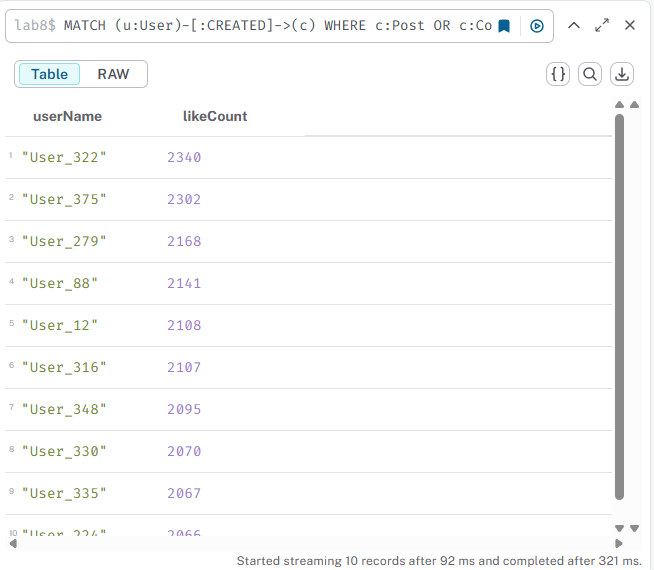


Рисунок 8.11 – Пример выполнения запроса 10

Составленные запросы были выполнены при различном ограничении узлов в исходном наборе. Затраты времени на выполнение при различном количестве узлов зафиксировали в таблице (таблица 8.3).

Таблица 8.3 – Затраты времени на выполнения запросов при первом запуске при различном количестве узлов

|  |  |  |  |
| --- | --- | --- | --- |
| № запроса | Общее количество узлов | | |
| >100 | >1000 | >10000 |
| 1 | 37мс | 162мс | 355мс |
| 2 | 16мс | 40мс | 135мс |
| 3 | 45мс | 175мс | 375мс |
| 4 | 49мс | 97мс | 162мс |
| 5 | 37мс | 60мс | 61мс |
| 6 | 32мс | 45мс | 64мс |
| 7 | 34мс | 24мс | 20мс |
| 8 | 24мс | 27мс | 32мс |
| 9 | 60мс | 870мс | 10973мс |
| 10 | 32мс | 210мс | 1061мс |

Neo4j при первом выполнении запроса инициализирует кэш и индексы из-за чего длительность выполнения запроса может быть больше. Последующие запросы выполняются значительно быстрее (таблица 8.4).

Таблица 8.4 – Средние затраты времени на выполнение запросов при последующих запусках при различных количествах узлов

|  |  |  |  |
| --- | --- | --- | --- |
| № запроса | Общее количество узлов | | |
| >1000 | >10000 | >100000 |
| 1 | 3мс | 85мс | 228мс |
| 2 | 2мс | 17мс | 79мс |
| 3 | 5мс | 87мс | 235мс |
| 4 | 3мс | 20мс | 54мс |
| 5 | 1мс | 6мс | 7мс |
| 6 | 2мс | 7мс | 12мс |
| 7 | 3мс | 3мс | 4мс |
| 8 | 2мс | 4мс | 6мс |
| 9 | 5мс | 740мс | 10273мс |
| 10 | 3мс | 105мс | 950мс |

Как видно по таблицам 8.3 – 8.4, время выполнения запросов может сильно зависеть от числа узлов в графе. При первом запуске, запросы выполняются дольше, так как происходит инициализация внутренних механизмов Neo4j. При последующих запусках время значительно сокращается. Также видно, что в запросах 9 и 10, где требуется работа с большим числом связей время возрастает сильнее.

**Вывод**

В ходе выполнения лабораторной работы было проведена исследование графовой базы данных на примере предметной области «Социальная сеть».

В ходе работы:

1. Была установлена и настроена среда Neo4j Desktop, создана база данных и подключение;

2. Разработана структура графа, включающая 5 типов узлов (User, Post, Comment, Group, Story) и 6 типов связей (FOLLOWS, FRIEND, LIKED, CREATED, MEMBER\_OF, COMMENTED\_ON), сгенерированы тестовые данные с использованием процедур APOC, обеспечивающих масштабируемость и устойчивость к переполнению памяти;

3. Выполнен широкий спектр запросов: поиск, сортировка, фильтрация, изменение и удаление данных, агрегатные функции;

4. Проведено измерение времени выполнения запросов при различных объёмах данных , выявлены особенности производительности.

Графовая модель оказалась особенно эффективной для представления сложных взаимосвязей между сущностями социальной сети.

Использование APOC-процедур позволило избежать проблем с производительностью при генерации больших объёмов данных.

Время выполнения запросов зависит от сложности запроса и количества узлов, особенно это заметно при первом запуске, когда инициализируются индексы и кэш.

Повторные запросы выполняются значительно быстрее, что демонстрирует преимущества Neo4j в сценариях с частым доступом к данным.

Запросы с агрегацией и множественными связями (например, запросы 9 и 10) требуют особого внимания к оптимизации, так как их время выполнения растёт значительно быстрее.

Графовая база данных Neo4j продемонстрировала высокую гибкость и выразительность при моделировании социальной сети. Она позволяет эффективно выполнять сложные запросы, отражающие реальные сценарии взаимодействия пользователей. Полученные результаты подтверждают целесообразность применения графовой модели в задачах, где важны отношения между сущностями.